

5

10

Appendix A

The E-Speak Architectural Specification Developer Release 3.03

09733027 120800

Chapter 1 Introduction

This document specifies the e-speak architecture. It defines the abstractions presented by the system and the components that implement those abstractions, and shows how the components interact to create useful services. The following companion documents are also available:

- E-speak *Programmers Guide* defines the interface for e-speak programmers and system developers building e-speak-enabled applications.
- E-speak *Installation Guide* shows how to install e-speak and how to run some simple applications.
- E-speak *Contributed Services* describes several sample applications included with the distributed software.
- E-speak *Tools Documentation* shows how to use tools provided for analyzing the system.

Vision

Computing with e-speak is a paradigm switch, aiming to bring a “just plug in, use the services you need, and pay per usage” model to computation, as opposed to the “install on your machine and pay per installation” model of computation prevalent today. E-speak is the infrastructure that realizes the vision of such a model. Instead of thinking of computing as some hardware you buy and the software you install on it, e-speak encourages you to think of computing as a set of services you access as needed.

The reality of computing today is that it is much more complex than a utility like the electric or water system. An immense variety of computing resources exists, both in type and in power, and a newer, faster, cheaper, or better resource is probably invented by the time you finish reading this sentence. This dynamism is a formidable challenge to interoperability.

At the same time, most of these resources are being connected to each other on a range of scales, from homes to companies to the entire globe. The hardware necessary to support such a computational utility is already available and getting better by the day. On the software front, though the Web has essentially achieved the status of a *data* utility, actual computation remains mainly confined to individual machines and operating systems.

E-speak enables a *computation* utility by interposing on and mediating every resource access in a process called *virtualization*. This broad abstraction yields a model where machines, ranging from a supercomputer to a beeper, can be looked at uniformly and can cooperate to provide and use services.

Goals

E-speak aims at enabling ubiquitous services over the network- making existing resources (e.g., files, printers, Java objects, or legacy applications) available as services, as well as lowering the barriers to providers of new services. The infrastructure's goal is to provide the basic building blocks for service creation, including:

- Secure access to resources and service
- Usage monitoring, billing, and access control
- Advertising and discovery of new services
- Mechanisms for negotiation to find the "best" service
- Independence of operating system, language, and device
- Ability to support large enterprise-wide, intra-enterprise, and global deployments

Architectural Philosophy

This document specifies the e-speak architecture. There are four key concepts:

- **Resource:** Any computational service, such as a file or a banking service, that is virtualized by e-speak.
- **Client:** An active entity that requests access to Resources or responds to such requests.
- **Protection domain:** The part of the e-speak environment visible to a Client.
- **Logical machine:** An active entity that performs the operations needed to implement e-speak.

E-speak is based on the following:

- All Resource access is mediated; e-speak sees all Resource requests.
- All Resource access is virtualized; e-speak maps between virtual and actual references.
- Names for Resources are shared by convention only; e-speak keeps a separate name space for each Client.

This document does not specify anything outside of the e-speak architecture. However, some implementation details are included to show some points. These sections are marked "informational."

Environment

E-speak is designed to work in a hostile, networked environment such as the Internet. It isolates service providers and their clients from an inherently insecure medium while allowing them to negotiate safely, form contracts, and exchange confidential information and services without fear of attack.

Intended Audience

This E-speak *Architecture Specification* describes the lower-level interfaces of e-speak for:

- Implementors of Client libraries to provide a higher level of abstraction for e-speak
- Implementors of utilities and tools to manage and manipulate e-speak
- Implementors of e-speak emulation routines that are used in the run-time environment for legacy applications
- Implementors of extensions to existing services and resources used by Clients
- System administrators who implement policies for security and resource lookup
- Those designing and building their own implementations of e-speak

Structure

This specification consists of the following major sections, in the order listed:

- An overview of the e-speak architecture
- A description of the data structures used by e-speak to describe Resources-Resource metadata
- The interfaces to the e-speak platform that are exposed as "Core-managed Resources"
- A description of Vocabularies, the mechanism for processing Resource descriptions to discover and match Resources to the Client's description of Resources needed
- The e-speak mechanisms used for access control
- The e-speak communication architecture
- The exceptions that can be generated by the e-speak platform

- The e-speak Event Service
- The e-speak management architecture
- The e-speak Respository used for storing Resource metadata (informational)
- A description of how localization is implemented (informational)
- A glossary of terms
- A brief description of probable future extensions to e-speak (informational)

Conventions

There are several document conventions worth noting:

- New terms are introduced in the document flow with italics.
- Programmatically visible architectural abstractions are written with the first letter of each word capitalized, such as Protection Domain.
- Logical names, method names, and other programmatic labels are written in Courier font.
- Even though e-speak is independent of the programming language, the specification uses Java syntax.
- Sections describing material outside of the architecture are shown with the word "Informational" in the chapter or section title.

008021-120800

Chapter 2 Architecture Overview

All system functionality and e-speak abstractions build on top of one single first-class entity in the e-speak architecture- a Resource. A Resource is a uniform description of active entities such as a service or passive entities such as hardware devices. Unlike most platforms, e-speak deals only with data about Resources, *metadata*, and not Resource-specific semantics. Thus, a file Resource within the e-speak environment is simply a description of the attributes of the file and how it can be accessed. The e-speak platform does not access the file directly. A Resource-specific handler that is attached to the e-speak platform receives messages from e-speak and directly accesses the Resource.

Access to e-speak is provided by the e-speak Service Interface (ESI). Client applications and Resource Handlers are linked with a library that provides this interface. The library communicates with the e-speak platform using the *Session Layer Security Protocol* (SLS). The e-speak platform mediates all Resource access. Every access to a Resource through e-speak involves two different sets of manipulations:

- 1 The e-speak platform uses its Resource descriptions for dynamic discovery of the most appropriate Resource, transparent access to remote Resources, and sending Events to management tools.
- 2 The Resource-specific handler directly accesses the Resource such as reading the disk blocks for a file.

E-speak treats all Resource accesses in exactly the same manner. This mediated yet uniform access is the design principle that allows the e-speak environment to accommodate any kind of Resource type flexibly, even Resources dynamically defined after the e-speak system has started.

The e-speak platform maintains an environment for each of its Clients, called a *Protection Domain*. A Protection Domain is analogous to a “home directory” in an operating system. It contains bindings to Resources created by the Client and e-speak keeps track of memory usage due to these bindings.

A single instance of the e-speak platform is called a *Logical Machine*. Figure 1 shows a single Logical Machine. There may be multiple Logical Machines on a single physical machine, or the components of a Logical Machine may be distributed across multiple machines. Logical Machines are independent entities that communicate using the Session Layer Security Protocol as shown in Figure 2.

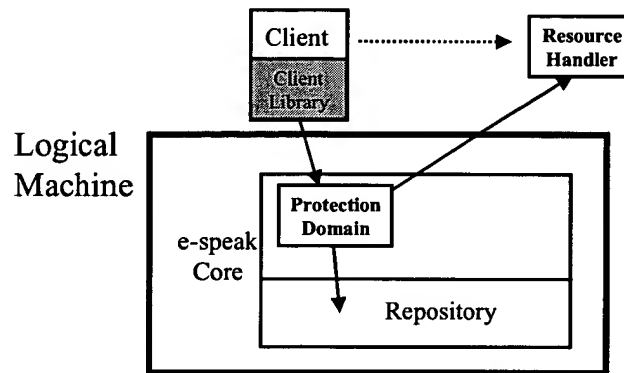


Figure 1 Resource access in e-speak

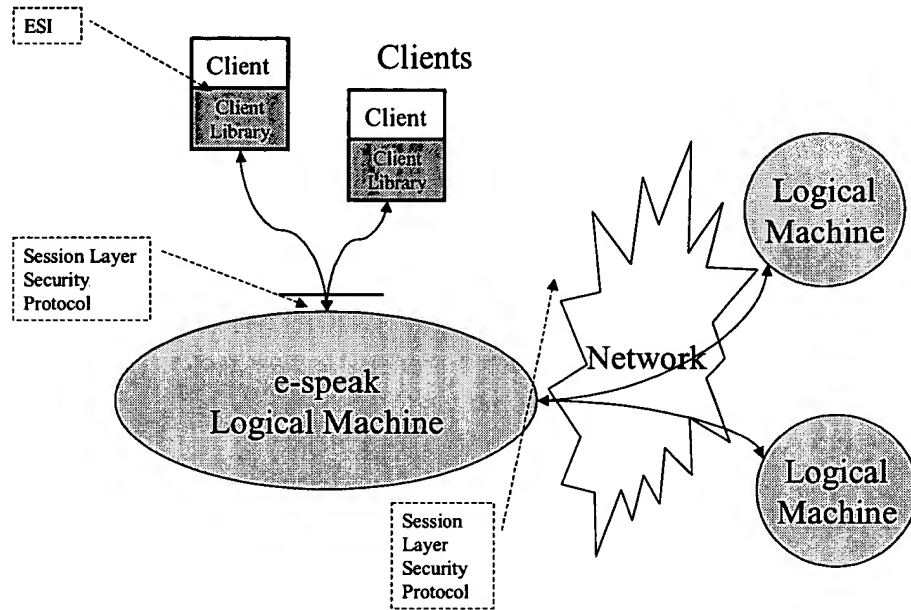


Figure 2 Communicating e-speak logical machines

Each e-speak Logical Machine has a single instance of the e-speak Core. All Resource access is through the Core that uses the Resource metadata to mediate and control each access. To access a Resource, a Client sends a message to the e-speak Core naming the Resource. The e-speak Core uses the Resource metadata to locate the Resource Handler and forwards the message to the Resource Handler that, in turn, physically accesses the Resource.

Although Figure 1 shows the Resource Handler being outside the Core (i.e., in a separate process), the handler for some Resources is the Core itself; these Resources are inside the Core and are called *Core-managed Resources*. E-speak Clients can manage and interact with the Core by sending messages to these Resources. For example, one kind of Core-managed Resource is a *Resource Factory*. When a Client wants to create a new Resource instance, it sends a message to the Resource Factory to register the Resource metadata with e-speak.

Logically, there are three categories of Resource access:

- The following sections outline the various components of the e-speak architecture and describe the steps in a service access.

- A set of Core-managed Resources inside the e-speak Core. The Core-managed services present the system functionality for managing the e-speak platform, including creating Protection Domains and their contents and managing Resource metadata.
- A Repository containing Resource metadata available to Clients of the Logical Machine. These are the metadata that the Core evaluates during any service access.
- A routing engine that routes all service access messages based on the contents of the metadata of Resources referred to in the parameters of the message. The implications of this are discussed below.

Resource Model

The Resource is a representation of an *e-service* within e-speak. Service providers register the metadata of their services (e-speak Resources) with the Core. This includes the information depicted in Table 1.

Table 1 Resource Model

Description	An attribute-based specification of the Resource
Vocabulary	The definition of the attributes and their types used in descriptions and lookups
Resource Handler Mailbox	The process/thread/task that handles the Resource
Contract	Denotes the Application Programming Interface (API) supported by the provider, including version and similar information
Resource mask, owner public key and service ID	Access control information
Private Resource-specific data	Data important to the provider of the Resource, such as the provider's internal name for the Resource. Not interpreted by e-speak.
Public Resource-specific data	Data important to the user of the Resource, such as a stub for invoking methods. Not interpreted by e-speak.

The Resource metadata is maintained in the mediating Core's Repository. All functionality presented through the Core must have metadata within the Core. This is true even for the functionality provided by the Core itself.

Metadata System

The e-speak metadata system is based on the following architectural and semantic entities:

- Vocabularies are created as first-class Core-managed services. Thus, the model includes a metalanguage for creating a whole range of vocabularies with which to describe services, much like that of XML. XML document type definitions (DTDs) can be handled through the e-speak *Vocabulary Builder*.

The representation of vocabularies as Resources ensures that they can be dynamically discovered and protected from illegal access, and that access to them is mediated as required, like any other service in e-speak. In the e-speak architecture, a created Vocabulary decides the validity of an attribute description provided by a registering service provider.

The Core-managed Vocabulary service also includes a matching engine that is used to match Resource descriptions available in the Repository with search requirements of Clients of e-speak.

- Attribute-based service descriptions are used by service providers as part of service registration. These attribute descriptions are sets of name-value pairs in a specific vocabulary. The Vocabulary is either one that the service provider previously created (using the Vocabulary Builder) or discovered through the discovery facilities provided by the e-speak Core services.
- *Search Recipes* are objects that hold a Client's recipe for discovering a Resource. The Core uses this to process the Resource discovery request. A Search Recipe specifies what Resources the Client is looking for, how the lookup should be done, and what should be done if multiple matches occur. The Search Recipe contains the predicates and a Repository view mechanism with which to constrain the search. A search predicate is constructed with a Vocabulary and a constraint string expressed in that Vocabulary. The predicate is expressed in a form based on the Object Management Group trader services constraint grammar.
- The operational realization of the metadata system includes support for including arbitrary advertising services as part of extended searches, arbiters to optimize matches found through the Core Repositories, and integration of

003021 20000600

Vocabulary translation services with the lookup/discovery process. Advertising services provide scalability to service lookup in e-speak by supplying a means to find Resources not registered in the local Repository. Arbiters are used to affect special purpose optimizations such as handling multiple hits in lookups. Translation services can be integrated with Core-managed Vocabulary services or created as external services, thus allowing for translation between Vocabularies.

Naming Model

The e-speak naming system is based on the following principles.

- E-speak Names are Universal Resource Locators (URLs).
- Name spaces are maintained in container Core-managed Resources called *Name Frames*. Name Frames can themselves contain other Name Frames, so each e-speak Core has a hierarchy of Name Frames beginning from its "root" Name Frame. By default, when the Client specifies the name of a service, the e-speak Core, starting with its root Name Frame, finds mapping between the name and a name unique to this Core. In addition, a client can specify a name beginning in the root Name Frame of another e-speak Core, by specifying the host in the e-speak Name.
- The e-speak Core provides the only valid reference to a service as a name in the Client-specific name space. This is like a virtual address of a service. The physical address of a service, the Core's name, is not a valid Client reference for a service.
- There are two ways for a Client to get a name for a service. First, another Client, application, or service provider can pass it the name. Second, a Client may obtain a per-Client name through a bind call that requires a Search Recipe as a parameter. The e-speak system (Core and Client libraries) looks up the name in local Repositories, known remote Repositories, and if necessary a global advertising service to locate the appropriate service and create a binding for the Client in its name space.

- Bindings in e-speak are objects that capture an algorithm. At their simplest, bindings may capture a Search Recipe. These bindings may be resolved and frozen to a specific Core name or names, resolved and cached, or simply resolved on each access. This gives the e-speak system an active naming model. Even when resolved, a Client reference may be bound to multiple Core names, which may be arbitrated prior to service access. This may be done by using a Client-specified arbitration service that picks one particular service from a list of services represented in a binding.

Access Control

E-speak security is based on a Public Key Infrastructure (PKI). Specifically it uses the Simple Public Key Infrastructure (SPKI) developed within the Internet Engineering Task Force.

All entities in e-speak (Resources and Cores) are identified by public keys. To authenticate an entity, we verify that it knows the private key corresponding to the given public key. No entity should ever intentionally share its private key or give anybody access to the private key.

Any entity can create a key-pair. Provided the private key is kept secret, the key-pair is unique to that entity. However, having a key-pair gives you no power in the system. It is necessary also to have certificates stating access rights issued to your public key.

In e-speak, access rights are stated in *attribute certificates*. So as well as the conventional use of certificates to bind a name to a public key (e.g. X.509), we also use certificates to bind access rights to public keys. This helps to have online access control databases or access control lists.

To decide whether to honor an incoming request a Resource Handler must decide if it has a certificate (or certificates) granting access rights for the request. If it finds such a certificate, it must verify that the sender of the request knows the private key corresponding to the public key in the certificate to which the access rights have been given (formally this is the subject of the certificate). It does this by a cryptographic protocol that is described in Chapter 6, "Communication".

Finally before honoring the request, the Resource Handler must verify that it trusts whoever issued the certificate. It does this by verifying that the certificate has been signed by an entity that it trusts. Resource Handlers do not trust all certificate issuer's equally. A Resource Handler can choose whether to trust a given certificate issuer and may restrict what access rights a given certificate issuer can issue.

Communication

E-speak uses a mailbox metaphor to describe the interactions between Clients and the Core. This metaphor does not imply that any actual messaging is required, only that the interfaces are defined in terms of mailboxes. Mailboxes consist of two forms: an *Outbox* and a Core-managed Resource called an *Inbox*.

When a Client wants to use a Resource, it constructs a message consisting of a message header and a payload and inserts the message in the Client's Outbox. The Outbox is connected to the Core, which processes the information in the message header. If there is no error, the Core extracts Resource specific metadata and security information from its Repository and inserts this in the message before forwarding the message to the designated Inbox. The Resource Handler reads the message header and the inserted payload to determine how to deal with the request.

The Resource specific metadata and security information inserted by the Core into a message can be used by the Resource Handler to determine how to process the message.

As each message is routed, the e-speak Core may generate events for logging and monitoring.

E-speak uses peer-to-peer communication. The Core has no concept of a reply message. If the Resource Handler needs to return a value to the Client, it must specify a Resource listing an Inbox connected to the Client in the handler field of its metadata. Hence, in replying to a message, the Resource Handler changes roles with the Client.

Session Layer Security Protocol

All messages exchanged between e-speak Cores and between e-speak Cores and Clients use the Session Layer Security protocol. This provides secure message passing between entities as well as unprotected message exchanges. Applications can choose whether to use secure message passing or not.

Session Layer Security protocol is designed to support e-speak mediation. E-speak mediation requires e-speak to modify certain parts of the message so that the message can be routed between endpoints and means there is no TCP connection between the endpoints. These requirements mean that existing protocols such as SSL (Secure Sockets Layer) or TLS (Transport Layer Security) are not suitable for end to end security in e-speak.

Session Layer Security protocol allows multiple secure sessions to be multiplexed over a single TCP connection. This means that two e-speak Cores can be connected via a single TCP connection with many Clients and have many different secure sessions to different e-speak Resources.

Session Layer Security protocol also supports tunnelling. During firewall traversal we might want the firewall to control the client access rights to the internal LAN for every packet. However, we might not want the firewall to see all the traffic in clear (therefore, losing the end-to-end security property). With Session Layer Security protocol we can nest a secure session inside another one, possibly with different end points, allowing us to achieve both goals simultaneously.

Session Layer Security protocol is designed to support SPKI for access control. It performs the negotiation of access rights that need to be proven represented by multiple SPKI certificates.

Session Layer Security supports the following encoding types for messages:

- **CLEAR_DATA:** The message is not encrypted or protected against modification.
- **PROTECTED_DATA:** The message is not encrypted but it is protected against modification.
- **SECURE_DATA:** The message is encrypted and protected against modification.

Session Layer Security has been designed to be independent of transport. However, for interoperability between e-speak Cores and interoperability between e-speak Cores and Clients, direct implementation over TCP is assumed. Other implementations are possible, including passing SLS messages over HTTP, or through shared memory.

Core to Core Communication

Communication between e-speak Cores uses the Session Layer Security protocol. Two core-managed Resources are used for remote communication between e-speak cores: the Connection Manager (for connection management) and Remote Resource Manager (for management of remote resource metadata).

The Connection Manager sets up the initial connection, manages it and closes it down when it is no longer needed. It requires the host name (or IP address) and port of the remote e-speak Core to set up a remote connection. The Connection Manager has a well known name: `es://<server>/CORE/ConnectionManager`. So given the host and port number (i.e. the `<serve>` part) a Connection Manager can negotiate with the remote Connection Manager to establish a connection between the two e-speak Cores. Once the two Connection Managers have established a connection, Cores exchange Resources with each other using their Remote Resource Managers for Resource export and import.

The Remote Resource Manager is responsible for managing metadata: importing and exporting resources from the remote e-speak core. The Remote Resource Manager on any given e-speak core is: `es://<server>/CORE/RemoteResourceManager`. So given that two Connection Managers have established a connection between two e-speak Cores, the Remote Resource Managers can communicate with each other to exchange Resources.

All resources can be exported by reference, in which case a copy of the metadata of the Resource is sent to the remote core. In addition certain Core-managed Resources can be exported by value, in which case a copy of the Resource is sent to the remote e-speak Core.

Resource import and export serves a number of purposes in e-speak.

- Resource discovery is made more efficient for local Clients, because a copy of the metadata is available locally.

- The lookup mechanism requires that a vocabulary is available locally for both Resource registration and lookup in that vocabulary. Using Resource export, a vocabulary can be defined once and exported to wherever it is needed.
- Name Frames can be defined containing a set of useful bindings (akin to an environment for a Client). Using Resource export, these Name Frames can be made available locally wherever a Client needs them. This can be particularly useful for mobile clients.

An End-to-End Example

When the Client on Logical Machine A sends a message to its Core for a Resource on Logical Machine B, the following steps take place (see Figure 3):

- 1 The Client constructs and Session Layer Security message setting the "to" address to the Name of the Remote Resource (e.g. `es://<host_for_core_B>/resource/foo`). The from address is set to the Name of the Client (e.g. `es://<host_for_core_A>/client/bar`). This message is sent using TCP to Core A, where it is placed in the Client's outbox.
- 2 A message handling thread in Core A, picks up the message and sends it to Core A's router. Core A's router determines that the message is for Core B. It checks that it has a connection with Core B and forwards the message to the relevant inbox.
- 3 A message handling thread on Core A picks the message up from the inbox and transmits it to Core B (via a TCP connection) where it is placed in the outbox for incoming messages from Core A.
- 4 The router on Core B resolves the Name for the "to address" in its root Name Frame. The resolved name is a binding to the Resource metadata.
- 5 Core B's router retrieves the Resource's metadata. This tells it to which inbox to send the message. It also extracts the Private Resource Specific data and various security information that is used by the Resource Handler to process the message.

- 6 A message handling thread picks the message up from the inbox and sends it to the Resource Handler through a TCP connection.
- 7 Any communication between the Resource Handler and the physical Resource is outside of the control of the e-speak Core.

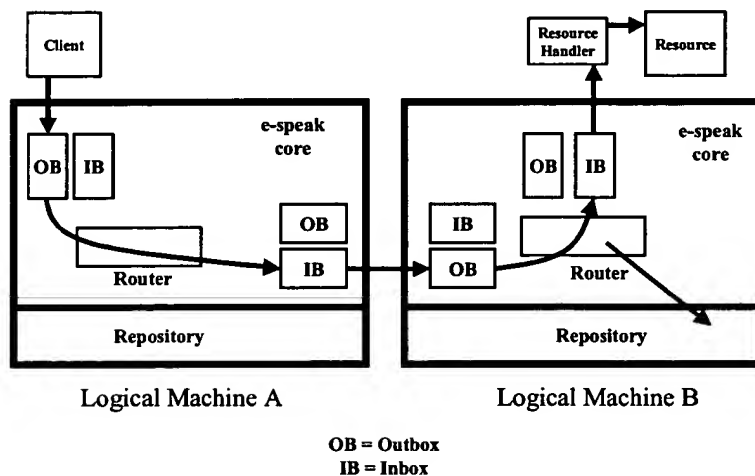


Figure 3 Distributed e-speak

The processing for any return message is similar, except the roles of the respective e-speak Cores are reversed (recall e-speak is asynchronous - the Core does not distinguish between request and reply messages when routing them).

The E-speak Service Interface (Informational)

An e-speak Client is an application running in its own address space that was written using an e-speak Service Interface (ESI). There is one ESI for each programming language supported. An example ESI is shown in Chapter 1 of the E-

speak *Programmer's Guide*. This provides a rich environment offering rapid, secure-service development, deployment, and management in a heterogeneous networked environment.

E-speak Services

E-speak has the following services: Event, management and advertising.

- The Event Service allows applications to collaborate by publishing Events and subscribing to Event distributors. The e-speak Core uses the Event Service to publish Events to the management service.
- The Management Services manage interconnecting sets of e-speak Cores, managing the distribution of metadata, and e-services registered as e-speak Resources.
- The Advertising Service is used for distributed Resource discovery in large-scale environments.

Standards

The e-speak platform builds upon and uses existing industry standards wherever possible. In some cases, integration with industry standards is under way or planned. The specific areas of integration include:

- Database access- The persistent back-end for the Repository uses Java Database Connectivity, thus making it possible to send Repository queries to almost any relational database.
- Advertising services- The Advertising Service back-end is provided by Lightweight Directory Access Protocol.
- Transport protocols: The ESIP messaging stack supports pluggable transports. TCP/IP, IrDA, WAP, and HTTP are all candidate transports.

- Service description: E-speak supports multiple different Vocabularies, including forthcoming support for XML and X.500 schemas.
- Component models: These models integrate the e-speak service abstraction with standard component models such as (Enterprise) Java Beans, CORBA, and COM+.
- Management protocols and standards: Support for SNMP, ARM, and DEN is planned.
- Languages: An E-speak library exists for Java, but e-speak has been designed to be language independent. Any language can be used to construct an Session Layer Security message which is all that is required to use e-speak.

Summary

E-speak presents a uniform service abstraction, mediated access, and manipulation of Resource metadata. This creates an open service model, allowing all kinds of digital functionality to be reasoned about through a common set of APIs. New service types and semantics can be dynamically modeled using the common service representation of an e-speak Resource.

The naming system provides active bindings and personal name spaces. The connection between Clients and Resources can be reasoned about and formed at run-time (upon each access if necessary) based on arbitrary search characteristics. Personalization of views and environments and hot-plug replacement of Resources all become possible.

The access control is based on a Public Key Infrastructure using attribute certificates for scalable distributed security. This is supported by the Session Layer Security protocol which allows messages to be protected against tampering, eavesdropping or replay. In addition the Session Layer Security protocol allows unprotected messages to be sent, should security not be needed. Session Layer also supports authenticated tunneling for efficient and secure firewall traversal.

The metadata system defines Vocabulary models as first-class entities in the system that can be reasoned about in the same manner as all other services. Translation and lookup through scalable advertising services are integrated into the model. Service location and discovery can thus seamlessly deal with a situation where the Client describes its requirement in an X.500 schema, while the service provider describes its service using an XML DTD.

The distribution model supports a flexible set of access methods. Thus, downloading printer drivers and the remote access of a file are equally well supported by the model. The separation of the infrastructure into interacting Logical Machines builds on the autonomous machine model provided by the Web.

These are the defining features of an open services platform. The collection of the capabilities discussed above creates an environment where services on the Internet can interact in a secure, dynamic, manageable way. The next chapter of the Internet (e-services) is being written, and e-speak helps us understand it.

008021 120800 09733027 29055760

Chapter 3 Resource Data, Searches & Vocabularies

Outline

E-speak Resources include all the entities called "Services" in the ESI, together with programs and data entities held in the Core to enable the use and management of Services. The Services are called "External Resources" in the Core software, and the aids to using them are called "Core-Managed Resources". The programs and data of an external Resource are not seen by the core, and are managed by an external Resource Handler. What is held in the core for any Resource is its *metadata*, consisting of:

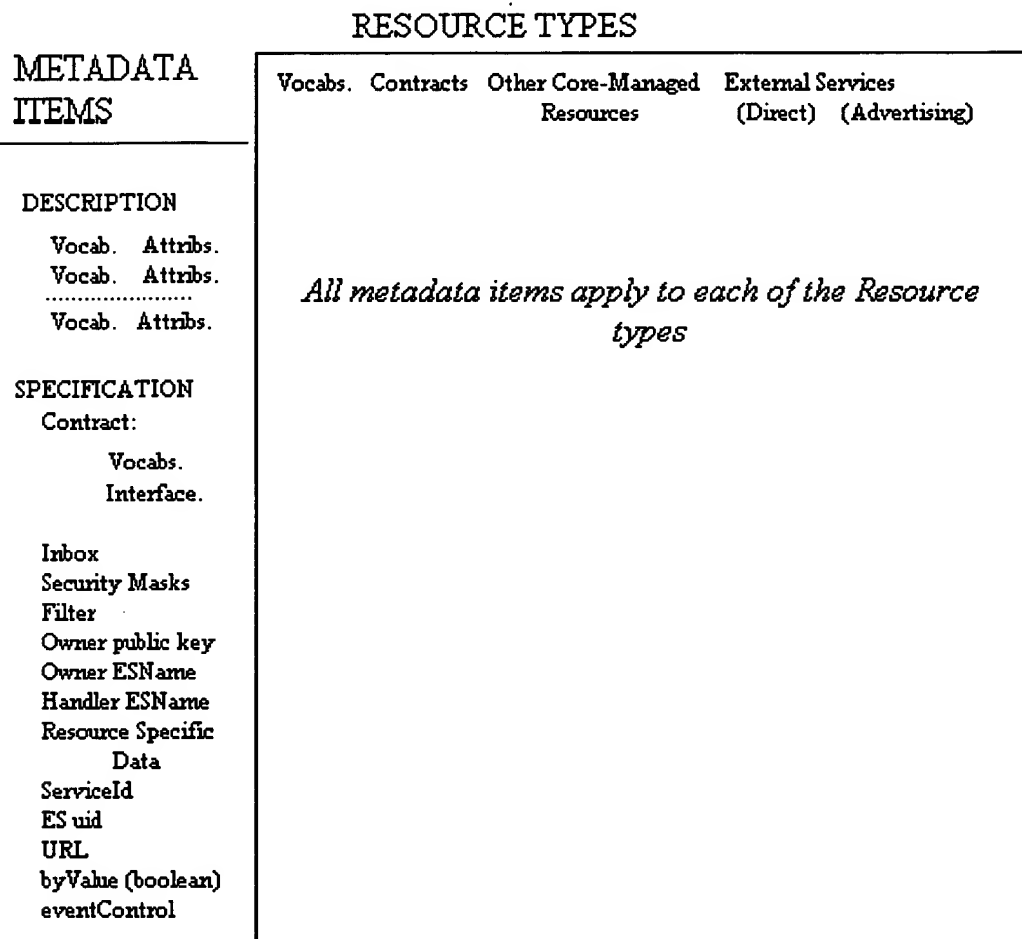
- *Resource Description*, allowing Clients to find the Resource.
- *Resource Specification*, enabling use of the Resource.

The types of Resource and a breakdown of the metadata that each Resource has are tabulated below. The terms will all be explained in this Chapter.

(One complication has been left out of the diagram and will not be discussed in this Chapter: the metadata for any Resource is itself an instance of a core-managed Resource class called *MetaResource*. *MetaResource* is discussed in Chapter 4, "Core-Managed Resources" under "Resource manipulation" on page 71 - the name of the interface used to act on Metadata.)

In the table below, *Vocabularies* and *Contracts* are separated from all the other Core-Managed Resources because they are both types of Resource and items of metadata. Advertising is distinguished from other (external) Resources because an ESI is likely to handle advertising differently from the creation of Services, as J-ESI does. The e-speak Core makes no distinction between advertising and other external Resources.

Figure 4 - Resource Types and Metadata



A Client registers a Resource by sending a message to the Core-Managed Resource, *ResourceFactory*, containing the metadata. If the registration succeeds, the Core returns a name bound to this Resource to the Inbox specified by the client. The metadata will be held under the same name in the Repository. The metadata comprises the whole of the information the Core uses to handle requests for the service and searches for it.

Resource Descriptions

The simplest Resource Description consists of a set of names and corresponding values, together with an ESName referring to a Vocabulary which specifies the kind of data under each name. For example, a translation Resource could have the description:

es://17.561.12.337:12356/vocabs/translation (Vocabulary reference)

name (String)	value (Value)	essential (boolean)
SourceLanguage	0x00 Japanese	true
TargetLanguage	0x0E English, Korean, Russian, Mandarin	false
InputFormat	0x00 Unicode	false
OutputFormats	0x0E ASCII, Unicode	false
Price	0x08 20000	false

This description is an instance of AttributeSet. The rows consisting of a name, a Value and the boolean essential are instances of Attribute. A ResourceDescription consists of one or more AttributeSet instances. Different parts of the description may have different Vocabularies, and if so must be expressed in different AttributeSets.

Vocabularies are explained at the end of this Chapter. Although each instance of Value has a value-type code (seen above), a Vocabulary is needed to define its significance. The "price" could be in Yen per kilobyte of input, for example.

The boolean `essential`, if true, means that the Resource will not be returned in response to a `SearchRecipe` that omits this Attribute name and value. This can be used to set passwords for the discovery of a Resource. (The use of the Resource is controlled by the security mechanisms in (Chapter 5, "Access Control").

Value class

Instances of `Value` have a 1-byte `tcode` and an Object `val`. The `tcode` indicates the type of `val`. It is one of these static final codes defined in the class:

```

STRING_TYPE_CODE = 0x00;
LONG_TYPE_CODE = 0x01;
DOUBLE_TYPE_CODE = 0x02;
BOOLEAN_TYPE_CODE = 0x03;
BIG_DECIMAL_TYPE_CODE = 0x04;
TIMESTAMP_TYPE_CODE = 0x05;
DATE_TYPE_CODE = 0x06;
TIME_TYPE_CODE = 0x07;
INTEGER_TYPE_CODE = 0x08;
FLOAT_TYPE_CODE = 0x09;
CHAR_TYPE_CODE = 0x0A;
BYTE_ARRAY_TYPE_CODE = 0x0B;
BYTE_TYPE_CODE = 0x0C;
SHORT_TYPE_CODE = 0x0D;
SET_TYPE_CODE = 0x0E;
NAMEDOBJECT_TYPE_CODE = 0x0F;
OTHER_TYPE_CODE = 0xFF;
INVALID_BASE_TYPE_CODE = 0xFF;

```

In all but the following cases, a `Value` is marshalled in e-speak serialization format. The exceptions are:

- `tCode = SET_TYPE_CODE`: `val` contains a set of values. This is not supported in the current release.
- `tCode = BIG_DECIMAL, DATE, TIME, TIMESTAMP, NAMEDOBJECT`: `val` is sent as a `String`. The first 4 types are taken from the java packages:
 - `java.math.BigDecimal`,
 - `java.sql.Date`,
 - `java.sql.Time`,
 - `java.sql.Timestamp`.

Resource Specifications

The ResourceSpecification class is defined below.

```
public class ResourceSpecification
{
    boolean byValue;
    ESName contract;
    FilterSpec filter;
    ADR metadataMask;
    ADR resourceMask;
    ADR ownerPublicKey;
    ADR ServiceId;
    ESMAP privateRSD; //Not exported if export by reference
    ESMAP publicRSD;
    ESName owner;      //Not exported
    ESName resourceHandler; //Not exported
    int eventControl;
    ESUID uid;
    String URL;
}
```

boolean byValue

This flag governs the export of the Resource to another Logical Machine. See (Chapter 6, "Communication"). If byValue is True, a copy of the Resource itself will be included with the Resource Specification and Resource Description exported. The copy of the Resource can then be used on the receiving platform. The Core will provide this copy for Core-managed Resources. Currently, there is no mechanism for providing copies of external Resources. They can only be exported "by Reference", and only used remotely.

ESName contract

The contract field is the name of the Contract Resource associated with the Resource. An e-speak Contract is not an agreement between parties, but a provision to make a Resource usable. It can contain:

- an interface which the Resource will implement.

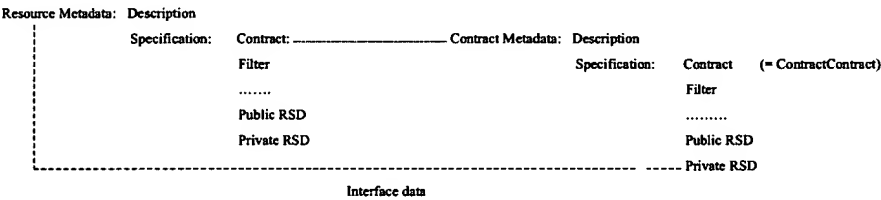
- one or more Vocabularies which can be used to frame queries in a search for Resources implementing that interface.¹

A search may be required because several Resources may implement the same interface.

A Resource cannot be registered without a valid contract field, so the Contract itself must have already been registered. The Contract given in the Specification of a Contract is ContractContract - what else? It is supplied by the Core.

Contract implementation (informational)

The current J-ESI creates ESContracts, containing an interface which has been generated by an Interface Definition Language (IDL) compiler, ES-IDL. The ESContract also contains Terms of Use, a Conversations scheme and the ESName of the interface. An ESContract creates a core Contract. All the interface information is put in the privateRSD field of this core Contract's metadata, in a format interpreted by J-ESI. The following diagram illustrates this.



A core Contract is constructed with an array of Vocabulary identifiers² and a ResourceType. The latter is one of several integer constants defined in the ResourceType class, and listed below. Contracts for all external Resources have a ResourceType of EXTERNAL_CODE, = 1000. Each Core-Managed Resource has an appropriately named ResourceType.

Current ResourceTypes:

```
static int INBOX_CODE = 0;
```

1 This feature may be discontinued

2 Vocabularies may be omitted from contracts in future releases

[illegible]

Contract Methods

Contract methods include:

- void register (ResourceDescription, ResourceSpecification, boolean persistence)
- RepositoryHandle[] getVocabularies()
- void addVocabulary(RepositoryHandle vocab)³
- String getResourceType()
- AbstractResource createResource(ResourceDescription, ResourceSpecification, boolean persistence, Object arg)
- void SendObject(MessageOutputStream) used to export a Contract
- Object ReceiveObject(MessageInputStream) used to import a contract

This is only a sample. A RepositoryHandle is a unique identifier of an item in a Repository (see Chapter 4, "Core-Managed Resources"). The last three methods provide a standard signature for creation, export and import of every kind of Resource.

FilterSpec filter

```
class FilterSpec{
    ESSet Vocabularies;
    String constraint;
}
```

The filter is a way to restrict discovery of the Resource to particular Clients or classes of client. The constraint String is made up of conditions which the Client making a search must satisfy. All the names of Attributes must be in one of the Vocabularies, and all the values must be of the type and within the range specified in that Vocabulary.

³ May be omitted in future releases

Filter implementation (informational)

The constraint String has to specify a source for each Attribute value, and the Vocabulary for it if there is more than one - because the Vocabularies might include different Attributes with the same name.

The source of Client details is commonly the Client's UserProfile - part of the AccountManager Resource (see Chapter 4, "Core-Managed Resources"). When this is so, the Attribute name in the constraint is prefixed \$user/. Otherwise, the Attribute can only be taken from the Resource Description. For example, if a ResourceDescription and a UserProfile both include the Attribute "State", the constraint might include "\$user/State = State".

If there are multiple Vocabularies, Attributes are preceded by the Vocabulary name and a colon. For instance, where Vocabularies called "home" and "workplace" are in the FilterSpec, the constraint could include "\$user/home:State = workplace:State". The second "State" here must be in the ResourceDescription.

ADR metadataMask

The metadataMask controls which operations manipulating the Resource's metadata will have security disabled - so no certificates or Message Authentication Code are required to invoke them. The interface name in the metadataMask will always be the ResourceManipulationInterface. The format of the metadata Masks is specified in Chapter 5, "Access Control" - Section "Disabling Security" on page 102".

ADR stands for Ascii Data Representation. It is an abstract class: one extension of it is ADRAtom, and other extensions implement the interface Tag. These are the real classes which a metadataMask can instantiate. The Tag or ADRAtom identifies the operations that are free of security restrictions. Tags are described in (Chapter 5, "Access Control").

ADR resourceMask

The resourceMask determines which operations supported by the Resource will have security disabled. The format of the Resource Masks is specified in Chapter 5, "Access Control" - Section "Disabling Security" on page 102".

ADR `ownerPublicKey`

This field contains the owner's public key or a hash of it. The format is specified in Chapter 5, "Access Control" - Section "SPKI BNF Formats" on page 109). The ADR extensions that can be used for `ownerPublicKey` are `PublicKey` and `Hash`.

ADR `serviceId`

This field contains the `serviceId` of the Resource. `ServiceIds` are defined in Chapter 5, "Access Control" - Section "Service Identity" on page 88". This field can be any extension of ADR; it usually implements `Tag`.

ESMap `publicRSD`

"RSD" stands for Resource Specific Data. `PublicRSD` is used by the Client registering the Resource to insert any information he wants potential users to know.

ESMap is an e-speak implementation of Hashtable. It consists of a series of Objects, each even-numbered Object being used as a key, and the following Object being the associated value. Both the key and value Objects are byte-arrays in the two RSD classes. ESMap is serialized as `ESArray` (see Chapter 6, "Communication" for the e-speak serialization format for `ESArray`). The e-speak convention for `ESArray` is that it consists of a sequence of pairs - preserving the key-value pairs of the ESMap.

In the two RSD classes, no duplicate keys, null keys or null values are allowed. Any of these raises an exception.

ESMap `privateRSD`

This field is used by the Resource Handler when a Client sends a message to the Resource. Access to the `privateRSD` is commonly confined to the Resource Handler, but permission can be granted to any task using the e-speak security mechanisms. The use of this field in a Contract for the interface implemented by the Resource has been noted. Otherwise the field is most often used to carry the Resource Handler's designation for the Resource.

ESName owner

The owner field is the ESName of the active Protection Domain of the Client that registered the Resource. A Protection Domain is a Core-Managed Resource corresponding to a user's home directory in Unix, or to a "folder" in J-ESI. (See [E-speak Programmer's Guide] and Chapter 4, "Core-Managed Resources".) This field can be changed to another Protection Domain by any Client that has authorization. It is an error if the ESName is not bound to any Protection Domain.

The `owner` and `resourceHandler` fields are not included when the `ResourceSpecification` is serialized for export (see Chapter 6, “Communication”). The `privateRSD` field is only included in an export serialization if the export is by value.

ESName ResourceHandler

This is the ESName of the inbox, to which messages sent to this Resource will be delivered. This field is always NULL for Core-managed Resources, and only for these. The Client that has connected to this Inbox will receive messages for this Resource. (The format of these messages is defined in Chapter 6, “Communication” - Section “Protocol Data Unit (PDU)”). It is an error if the ResourceHandler ESName specified by the Client is not bound to an Inbox.

int eventControl

If `eventControl` is non-zero, then whenever the Resource metadata (the Resource Description or the Resource Specification) is changed, an Event will be published to the Core's Event distributor.

ESUID

```
public class ESUID
{
    byte[] UniqueId;
}
```


An ESUID contains a byte array that is up to 64 bytes long. The ESUID of a Resource is guaranteed unique to a very high probability. As the URL and the ServiceId also identify the Resource, it is not certain that this field will be retained. Currently it is used in some Core programs.

String URL

This field is the ESName (represented as a String) by which the registering entity refers to the Resource. It is an ESName (URL) which others can use to access the Resource.

Searches

Search Context (Informational)

A search is initiated by a Client. In the current J-ESI, the Client can create a Finder of one of three classes - ESVocabularyFinder, ESContractFinder or ESServiceFinder. Each has a `find (ESQuery)` method, where the `ESQuery` expresses the attributes the Client needs. Clients providing a Service are likely to use the `ESVocabularyFinder`, to obtain a suitable standard Vocabulary to describe the Service. This can make it more widely visible. Either providers or users may search for a standard Contract. A user may then:

- Construct an `ESServiceFinder`, stating the Contract or its interface.
- Use Vocabularies obtained from the Contract⁴ or an `ESVocabularyFinder` to frame an `ESQuery` argument
- Call `ESServiceFinder.find` with that `ESQuery` to discover the Service he wants - in one or more stages.

⁴ Contracts may not continue to hold vocabularies

If the J-ESI find is successful, a stub for the Resource is returned to the client. He can then call the Resource as if it were on the same platform. The stub will generate, serialize and send the PDU messages through the core needed to invoke the Vocabulary, Contract or Service methods (see Chapter 6, "Communication").

All three J-ESI Finders use the same core Finder Resource class, and there is no distinction in the core software between finding a Vocabulary, Contract or external Resource. The ESQuery is represented by a *SearchRecipe*, described below. The information content of the SearchRecipe will be the same as that of the ESQuery.

Finder Resource

This is a Core-Managed Resource to carry out a Client's find() command. It provides for searching in several stages, if many Resources satisfy the query.

Initial Search

An initial search is carried out by this method:

```
interface FinderInterface {
    FinderResults find(SearchRecipe recipe, int maxToFind)
    throws ESInvocationException, LookupFailedException;
```

The argument `maxToFind` is the maximum number of results to return. If it is set to 0 then the request is only to know if there are any search results - not what they are. If it is set to -1 the method returns all results found. There is a field in *SearchRecipe* which serves the same general purpose, so `maxToFind` may be discontinued. We refer below to "the limit" however it is set.

The returned *FinderResults* has these fields and public get --() methods to retrieve each of them:

```
class FinderResults{
    private ESname[] esnames;
    private int [] serviceIds;
    private FinderContext context;
```

The first two fields contain ESNames and serviceIds of Resources matching the *SearchRecipe*. The size of each array will be either the number of Resources which match, or the limit - whichever is least. A *FinderResults* method, `boolean hasMoreResults()` returns true if there are more Resources than the limit

Follow-on searches

If `hasMoreResults()` is true, then another batch, again up to the limit, can be obtained from a follow-on search, using the opaque `FinderContext` byte array context from the previous `FinderResults`. (The `getContext()` method obtains context, but is not available for Clients to use directly.) The follow-on method is:

```
FinderResults find(FinderContext context)
throws ESInvocationException, LookupFailedException;
}
class FinderContext{
byte[] queryContext
```

Finder details:

- The `LookupFailedException` is raised when there was an error in the Core during the search.
- When the limit is 0, (meaning one only wants to know if any Resources match the SearchRecipe) the initial search will return a non-null `FinderResults` object if there are some. Call this "outcome". The code:

```
ESNames[] outray = outcome.getESNames();
```

will yield an array of length 1, but `outray[0]` will be null. If there are no results, outcome is null.

SearchRecipe

A `SearchRecipe` is the expression of attributes a Resource must have to satisfy a Client's needs. Instances of the class are constructed with the following arguments:

- `ESSet` vocabularies - a set of `ESNames` of the vocabularies holding the attributes used in the constraint and preferences. The registered names of these vocabularies are prefixed to the attribute-names in the constraint and preferences if more than one vocabulary is used - otherwise there could be confusion between homonymous attributes.

- **String constraint** - consisting of attribute-values, relational operators and logical operators, which must be satisfied by the corresponding attributes in a ResourceDescription, for the Resource to qualify. For instance: "price LE 20000 AND maker's_name EQ Hewlett-Packard" could in principle be part of a constraint. (The current syntax used is different - see below.)
- **An optional ESArray preferences** - an array of Preference objects, described below, which can be used to rank Resources in an order of preference, if more than one Resource satisfies constraint.
- **An optional int arbitration** - limiting the number of resources to return.
- **An optional ESName repositoryView** - a set of RepositoryHandles delimiting the ResourceDescriptions which will be checked against the constraint. (See Chapter 4, "Core-Managed Resources").

SearchRecipe Context (Informational)

Clients specify SearchRecipes only indirectly, using an e-speak API. The Core architecture only specifies the data types of the SearchRecipe fields, not their internal syntax or meaning. However, implementers may need to know about the latter, which is that of the corresponding fields in a Client's ESQuery.

Constraint field

The syntax of the constraint field conforms to the OMG Trader Services Constraint Language, except for the means of testing multi-valued attributes. See [ESRL Spec V4.1] and [CORBA services Document 12].

Preferences field

The elements of the ESArray preferences are instances of the Preference class:

```
Class Preference
{
    final static int MIN= 1;
    final static int MAX= 2;
    final static int WITH= 3;
    private int type;
    private String expression;
    private String weight;
    .....
}
```

Examples (omitting vocabulary qualifiers, and not using a formal syntax):

MIN "Price * Mileage"

WITH "Color != Green" "2"

Assume these Preferences, in the order given, are used to order a set of car "Resources" which have already passed the constraint.

- ## Arbitration field

The field may be negative, with values defined in:

```

class ArbitrationPolicy
{
    public static final int ALL= -1;
    // To return all resources passing the constraint
    public static final int ANY= -2;
    // To return 1 resource - the most preferred if there are
    preferences
    public static final int NEGOTIATE= -3;
    // Reserved to invoke an arbitrator resource - not currently used
}

```

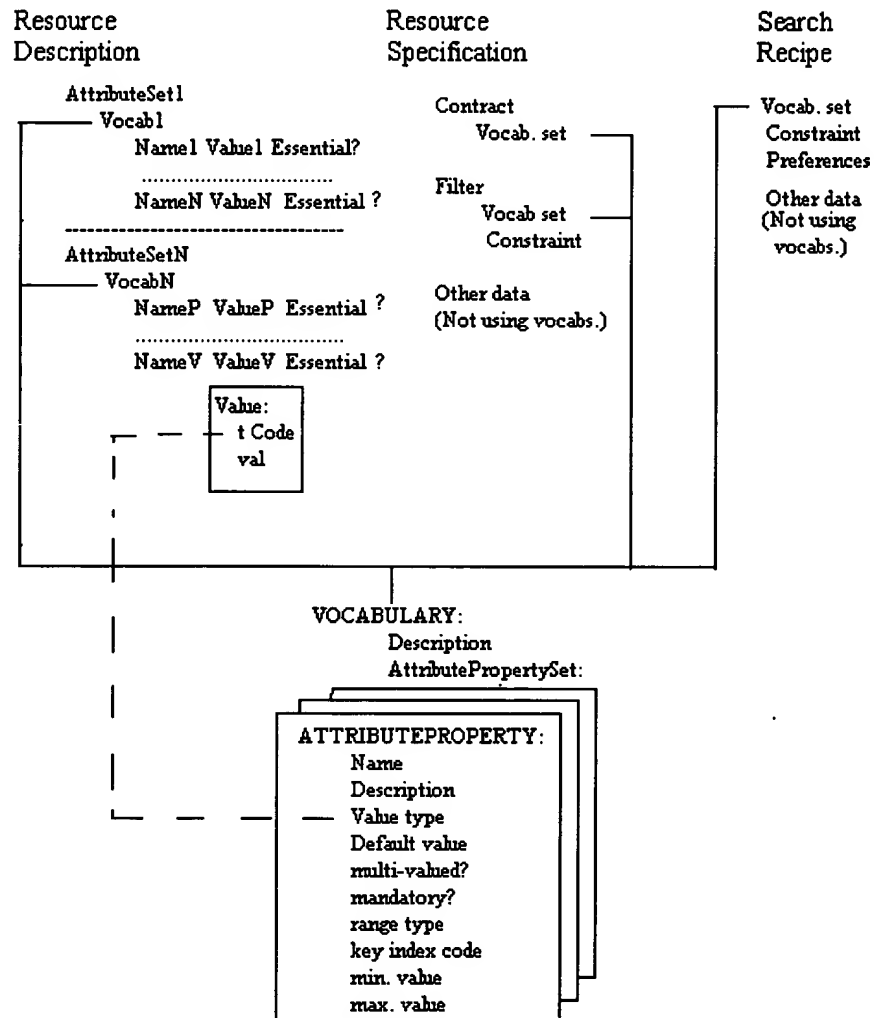
Vocabularies

An e-speak Vocabulary is a description of terms used in a Resource Description, in parts of a Resource Specification or in a Search Recipe.

Formally, a Vocabulary is an `AttributePropertySet`, as described below, with a description String. It is itself a (Core-Managed) Resource, with a `ResourceDescription` and Specification, and it can be searched for. The `ResourceDescription` of a Vocabulary must itself have a Vocabulary.....To end the recursion, the e-speak Core will ship with a *Base Vocabulary* preloaded. The Base Vocabulary will always be in the Core and accessible to all clients. Descriptions that don't use any other Vocabulary use the Base Vocabulary. The Core also supplies a Vocabulary Contract, for the `ResourceSpecification` of a Vocabulary.

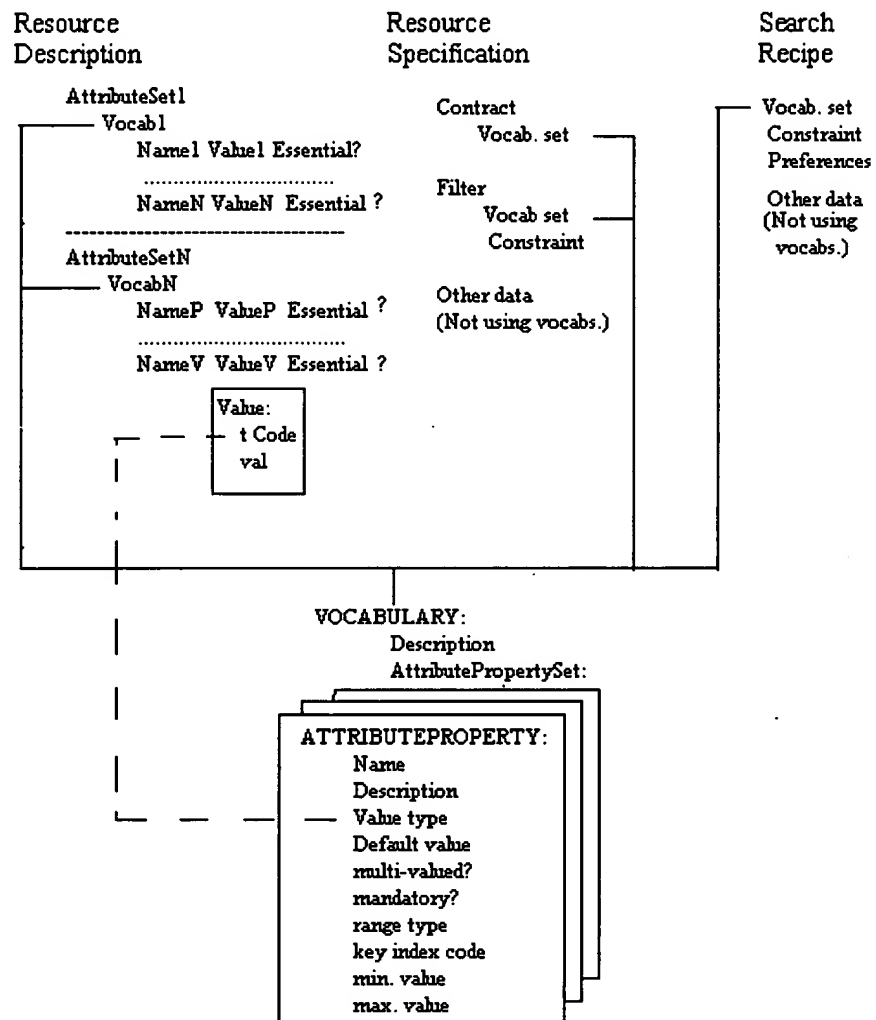
The following diagram summarizes the use and make-up of Vocabularies.

Figure 5 Vocabularies and their Uses



Notes on Preceding Figure:

- Items followed by a '?', such as "essential ?" are boolean.



- The dashed line from t Code to value type indicates a one-to-one correspondence. For example, if an Attribute "Price" in an AttributeSet has the tCode INTEGER_TYPE_CODE (0x08), the Vocabulary of that AttributeSet must have the ValueType "Integer" in the entry for "Price".

Vocabulary Context (Informational)

Vocabularies are essential both to describe and specify Resources, and to discover them. Any Client can make up and register his own Vocabularies, but if most of them did so, we'd have a tower of Babel, or worse. It is expected that standardization bodies or Service associations will develop all the significant Vocabularies, advertise them and control them using Access Control (Chapter 5, "Access Control").

Vocabulary class and methods

The Vocabulary class is outlined below:

```
class Vocabulary
{
    private String description;
    private AttributePropertySet props;

    String getDescription()
    throws ESInvocationException;

    AttributePropertySet getProperties()
    throws ESInvocationException;

    boolean mutateProperties(AttributePropertySet props)
    throws ESInvocationException QuotaExhaustedException,
    StaleEntryAccessException;
    // Exchanges current AttributePropertySet for props - returns
    true if any difference
    boolean isLegalSet(AttributeSet ast)
    // Checks if ast is legal under this Vocabulary, using
    AttributePropertySet.isLegalSet()
    public Vocabulary release()
    public void finalize()
    // Both of these take the Vocabulary out of use
}
```

The standard createResource(), sendObject() and receiveObject() methods are not listed.

Copyright © 1997-2000 by Sun Microsystems, Inc.

AttributePropertySet (APS) Methods

The following methods of AttributePropertySet are used in building a Vocabulary or registering a Resource

- Object `addAttributeProperty(AttributeProperty ap)` - this is the way an APS is built up.
- boolean `isLegalSet(AttributeSet attrs)` - this is called by Vocabulary. `isLegalSet()` to test whether `attrs` is valid for this APS. The tests made are:
 - Are all mandatory attributes included in `attrs`?
 - Is every attribute in `attrs` in this APS?
 - Do all the tCodes of the Values in `attrs` agree with the corresponding ValueType in the APS?
 - Where there is a range defined in the APS, is the Value of the corresponding Attribute in `attrs` within that range?
- AttributePropertySet `getMandatoryAttributes()` - used by the preceding method, and maybe by others.

Attribute Properties

An APS is an ESMap: it consists of a paired name and AttributeProperty instance for each Attribute in the Vocabulary. The name is duplicated within the AttributeProperty. The following table shows the fields of AttributeProperty.

Table 2 Components of an attribute property

Type	Field	Meaning
String	attrName	Attribute name
String	description	Human-readable description
ValueType	attrValueType	See Table 3 for encoding
Value	defaultValue	Default value
String	definition	Expression to evaluate for a dynamic attribute
boolean	multiValued	True if multiple values
boolean	mandatory	The attribute must be specified if mandatory is True
int	rangeKind	0 NO_RANGE 1 LEFT_RANGE 2 RIGHT_RANGE 3 FULL_RANGE
int	keyIndexType	Use for repository lookup: see below
double	minRange	Smallest allowed value
double	maxRange	Largest allowed value

- **defaultValue**: If the attribute is absent from an AttributeSet A, and is not mandatory, the XXXXXX will insert it in A with defaultValue.
- **definition** : Dynamic attributes are not supported in the current release.

- **multivalued**: If this is true, an ESSet of values is expected, and the **defaultValue** should be such a set. (See the e-speak serialization format in Chapter 6, "Communication" for the definition of ESSet).
- **mandatory**: If true, all **AttributeSets** under this Vocabulary must include the **Attribute** named. You won't be able to register a **Resource** whose description uses the Vocabulary, if the **AttributeSet** lacks this **Attribute**.
- **rangeKind**: specifies what kind of range-checking will be done on the **Attribute's** value, in relation to **minRange** and/or **maxRange**. The capitalized alternatives are integer constants defined (static final) for the class:
 - **NO_RANGE** - There is no restriction, whatever **minRange** and **maxRange** may be.
 - **LEFT_RANGE** - Values must be \geq **minRange**
 - **RIGHT_RANGE** - Value must be \leq **maxRange**
 - **FULL_RANGE** - Both **LEFT_RANGE** and **RIGHT_RANGE** rules apply.
- **keyIndexType** : To support lookup in a repository based on a Database Management System (DBMS). Valid values of **keyIndexType** are: **NO_INDEX** (0), **HASH_INDEX** (1) and **TREE_INDEX** (2). If the value is **HASH_INDEX** or **TREE_INDEX** the attribute will be used for index look-up by the DBMS. This is discussed further in Chapter 10, "Repository (Informational)".

Class ValueType

Instances of this class have the private fields:

```
String typeName;      // Must be one of the designators in Table 3
String description;  // Human-readable description
String matcher;      // An applicable relation, such as "islessThan"
int baseTypeCode;    // Equal to tCode in Values, except for invalid Type
```

typeName must be one of the Designators in the following table. These are defined (as static final String) in the **ValueType** class. Each corresponds to one of the allowable **tCodes** in class **Value**. The Matching rules and Operations are used to check the validity of expressions in a **SearchRecipe** or filter.

Table 3 Supported value types

Data type	Designator	Matching rules	Operations
Big decimal	"BigDecimal"	eq, ne, lt, le, gt, ge	+, -, *, /
Boolean	"Boolean"	eq, ne	AND, OR
Byte	"Byte"	eq, ne	
Byte array	"ByteArray"	eq, ne	
Char	"Char"	eq, ne	+ (concatenate, returns String)
Date	"Date"	eq, ne, lt, le, gt, ge	
Double	"Double" ¹	eq, ne, lt, le, gt, ge	+, -, *, /
Float	"Float"	eq, ne, lt, le, gt, ge	+, -, *, /
Int	"Integer"	eq, ne, lt, le, gt, ge	+, -, *, /
Long	"Long"	eq, ne, lt, le, gt, ge	+, -, *, /
Object	"NamedObject"		
Short	"Short"	eq, ne, lt, le, gt, ge	+, -, *, /
String	"String"	eq, ne	+ (concatenate)
Time	"Time"	eq, ne, lt, le, gt, ge	
Time stamp	"Timestamp"	eq, ne, lt, le, gt, ge	

1. Equality tests with a Float or Double may give "false" unexpectedly

Base Vocabulary

The Base Vocabulary available at system start-up includes the attributes and value types shown in Table 4.

Table 4 Base Vocabulary definition

Attribute name	Value type	Comments
Name	String	
Type	String	
ResourceSubtype	String	
ESGroup	String	
ESCategory	String	
Description	String	
KeyWords	String	Multivalued
Version	String	
ESDate	Date	"YYYY-MM-DD"
ESTime	Time	"HH:MM:SS"
ESTimeStamp	TimeStamp	"YYYY-MM-DD HH:MM:SS.FFFFFFFF"
HashAlgorithm	String	
HashCode	BigDecimal	To authenticate contents

The hash algorithm is specified using well-known names, for example, MD5.

Base Account Vocabulary

The Base Account Vocabulary is also available at startup. It is used for discovering user accounts.

Table 5 Base Account Vocabulary

Attribute name	Value type
UserName	String
UserInfo	String
UserType	String
UserLocation	String
UserESURL	String

The above attributes match the fields defined in AuthInfo and UserProfile. For more information, see "The Account Manager Resource" section in Chapter 4, "Core-Managed Resources."

The description string for the Base Account Vocabulary is: "E-speak base user vocabulary".

Translators (Informational)

The interoperation of different Vocabularies may be supported through *Vocabulary Translators*. The translator would map attributes from one Vocabulary into another, but there is no direct linkage between a Translator Resource and any Vocabulary Resource. A translator service is not part of the e-speak architecture; it would be an external Resource.

The translator envisaged would implement:

```
ESName[] [2] getVocabularyPairs();
```

```
boolean isCompatible(Vocabulary vocabulary1,
Vocabulary vocabulary2)
```

```
SearchRecipe translate(SearchRecipe s,
Vocabulary v2;
```

References

E-Speak Programmer's Guide Beta 3.0, June 2000

Developer Release X.03.03.00, September 2000

Chapter 4 Core-Managed Resources

Clients interact with the e-speak Core by sending messages to Core-managed Resources. For example, the Resource Factory is used to register new Resource metadata. This Chapter lists all the core-managed resources and describes those which are not described in other Chapters. It also describes the internal state that is passed if the Core-managed Resource is exported by value to another Logical Machine.

Conventions

All the methods described in this Chapter throw `ESInvocationException` (see Chapter 7, "Exceptions"), the base class for exceptions thrown by the e-speak Core to the Client during message processing.

Each class, of which instances can be exported by value, starts with a list of static declarations. Each declaration contains a permissible content of the payload in a PDU (see) requesting the core to invoke a method in that class.

The Account Manager Resource

The Account Manager Resource is for managing user accounts on an e-speak Core. A user account contains information about the user including its PSE (Private Security Environment). This enables a user to authenticate to the Account Manager (via userid, password) and to retrieve its PSE. In the current implementation it will

then need a passphrase to unlock the PSE to access its key material. The placing of the PSE under the Account Manager Resource does not implement the SPKI requirement for absolute security of private keys (see Chapter 5, "Access Control").

User Profile

The class `UserProfile` defines the basic information stored by the Account Manager for each user.

```
class UserProfile{
    AuthInfo authInfo;
    String userESURL;
    String userInformation;
    String userType;
    ProfileAttributeSet preferences;
    byte[] pse;
}
```

User Identity

The class `AuthInfo` defines the basic information used by the Account manager to identify a user

```
class AuthInfo{
    String userName;
    String passPhrase;
    String homeAddress;
}
```

The home address indicates the "home e-speak Core of a user" in host:portNumber format. An example is:

```
myhost.myCo.com:1234
```

User's Account

The `userESURL` is the ESName of the user's Account Resource. This is a Protection Domain. This ESName is bound to the user's Protection Domain in a Name Frame created by the Account Manager. Note that this ESName also includes the host and portNumber of the user's "home e-speak Core". An example of a `userESURL` is:

```
es://myhost.myco.com:12345/Core/AccountManager/
myhost.myco.com:1234/myName
```

When the Account is registered a Protection Domain is created and registered using the Base Account Vocabulary (see in Chapter 3, "Resource Data, Searches & Vocabularies") with attributes from `AuthInfo` and `UserProfile`. This means the Account Resource (Protection Domain) can be discovered using attribute-based `find()`, just like any other e-speak Resource.

User Type and Information

`UserType` and `UserInfo` are arbitrary strings that can be assigned by an application. These are defined in the BaseAccount Vocabulary, so they can be used to find Users.

Private Secure Environment (PSE)

The byte array `pse` is opaque, not interpreted by the e-speak Core.

Preferences

The `<ProfileAttributeSet preferences>` field is a set of name, value pairs defined as follows.

```
class ProfileAttributeSet{
  AttributeSet attrs;
  String format;
}
```

If the format string is set to "VOCAB", the `AttributeSet attrs` will be defined in a vocabulary specified in the `attrVocab` field of the `AttributeSet` (see Chapter 3, "Resource Data, Searches & Vocabularies"). Otherwise the format string will be set to "SIMPLE" and `attrs` will contain an arbitrary set of name-value pairs, not necessarily valid in any vocabulary.

The `ProfileAttributeSet` contains secret information. The intent is that this information should not be visible to any application other than the one that registered the account..

Account Manager methods

The following methods can be specified in the method field of a MessageForResource PDU with AccountManagerInterface in the interface field.

```
public interface AccountManagerInterface {

    public String registerUser(UserProfile up)
    throws PermissionDeniedException, StaleEntryAccessException,
    NameNotFoundException;

    public boolean unregisterUser(AuthInfo authInfo,
    String accountName)
    throws PermissionDeniedException, StaleEntryAccessException,
    NameNotFoundException;

    public boolean authenticateUser(AuthInfo authInfo)
    throws PermissionDeniedException, StaleEntryAccessException,
    NameNotFoundException;

    public UserProfile getUserProfile(AuthInfo authInfo,
    String accountName)
    throws PermissionDeniedException, StaleEntryAccessException,
    NameNotFoundException;

    public boolean setUserProfile(AuthInfo authInfo, UserProfile up)
    throws PermissionDeniedException, StaleEntryAccessException,
    NameNotFoundException;

    public String[] getAllUsers()
    throws ESInvocationException;

    public boolean addDescription(AuthInfo authInfo,
    String accountName, AttributeSet as)
    throws ESInvocationException;

    public String getUserESURL(String accountName)
    throws ESInvocationException;
}
```

The function getAllUsers returns a list of the ESNames (in stringified form) of the Account Resource (Protection Domains) of all registered users.

09733027-120800

The function `addDescription` is used for adding a new `AttributeSet` to the user's Account Resource (Protection Domain). This can be in any vocabulary, not just the Base Account Vocabulary.

The `accountName` parameter in `getUserProfile` and `getUserESURL` must match the `userName` in the `AuthInfo` of the intended account.

The function `getUserESURL` returns a `String` corresponding to the `ESNames` (URLs) of the user's Account Resource (Protection Domain).

Connection manager

The connection manager is described in Chapter 6, "Communication".

Core management resource

The core management Resource is deprecated in the current release. It may later be part of a Management API, and/or be changed. The current methods are:

```
interface CoreManagementInterface extends ManagedServiceIntf{
    int ping(int pingValue)
    throws ESInvocationException;
    ESName[] getClientConnections()
    throws ESInvocationException;
    boolean stopServingOutbox(ESName ProtectionDomain)
    throws ESInvocationException;
    boolean stopServingInbox(ESName Inbox)
    throws ESInvocationException;
    boolean startServingOutbox(ESName ProtectionDomain)
    throws ESInvocationException;
    boolean startServingInbox(ESName Inbox)
    throws ESInvocationException;
    boolean removeProtectionDomain(ESName ProtectionDomain)
    throws ESInvocationException;
    boolean denyNewClientSessions()
    throws ESInvocationException;
    boolean acceptNewClientSessions()
```

```

throws ESInvocationException;
    long getTotalMemory()
throws ESInvocationException;
    long getFreeMemory()
throws ESInvocationException;
    void startJVMGC()
throws ESInvocationException;
    void stopJVMGC()
throws ESInvocationException;
    void setJVMGCInterval(int millis)
throws ESInvocationException;
    int getJVMGCInterval()
throws ESInvocationException;
    boolean isJVMGCRunning()
throws ESInvocationException;
    void startScavenger()
throws ESInvocationException;
    void stopScavenger()
throws ESInvocationException;
    void setStatsNum(int num)
throws ESInvocationException;
    ESArray1 getScavengerStats()
throws ESInvocationException;
}

```

The Core Management Resource provides a way for a client to manage its core. By invoking the Resource on another core it can use the same methods on that too. (The client must have appropriate certificates in any case.) The Core Management Resource is itself a Core-managed resource: it implements the interface `ManagedServiceIntf` described in Chapter 9, "Management".

The method `ping` checks that the core is up and returns the value specified

The method `getClientConnections` returns a list of protection domains that are currently being used.

The methods `stopServingOutbox` and `startServingOutbox` tell the e-speak core to stop or start serving the outbox associated with the protection domain specified.

The methods `stopServingInbox` and `startServingInbox` tell the e-speak Core to stop or start serving messages to the Inbox specified.

¹ The current implementation returns an instance of the Java Vector class.

The method `removeProtectionDomain` removes the Protection Domain specified. Any client using the Protection Domain is disconnected and any Resources contained in the Protection Domain are deregistered.

The methods `denyNewClientSessions` and `acceptNewClientSessions` tells the e-speak core to stop or start accepting new connections from clients.

JVM management methods

The following methods are specific to e-speak Cores implemented in Java. Some e-speak Cores may not implement these methods: if so, the method will return a `MethodNotImplemented` exception.

The methods `getFreeMemory` and `getTotalMemory` get the free memory or total memory in the e-speak Core's Java Virtual Machine (JVM).

The methods `startJVMGC()` and `stopJVMGC()` start and stop the e-speak Core's JVM garbage collector.

The methods `setJVMGCInterval` and `getJVMGCInterval()` set and get in milliseconds the interval between runs of the JVM garbage collector.

The method `isJVMGCRunning()` returns true if the JVM garbage collector is running.

Scavenger management methods

The current implementation of the e-speak Core has a scavenger that looks for resources in the repository that are no longer valid and removes them. It can remove resources even if references to them still exist, unlike the JVM garbage collector. Examples of resource that may no longer be valid include the following.

- Resources registered in a Protection Domain that has been removed.
- Resources imported from another e-speak Core after the connection to that Core is closed.

Some e-speak Cores may not implement these methods: if so the method will return a `MethodNotImplemented` exception.

The methods `startScavenger` and `stopScavenger` enable and disable the scavenger from running.


```
class ScavengerStats
{
    Int runNo;
    Long timeElapsed;
    Int numInspected;
    Int numCollected;
    Int totalNumInspected;
    Int totalNumCollected;
    String phase;
}
```

The scavenger keeps statistics for a certain number of runs. This is set by method `setStatsNum` in the `CoreManagementInterface`. The method `getScavengerStats` returns an `ESArray` of containing an instances of `ScavengerStats` in each element. (The current implementation returns an instance of the Java `Vector` class.)

Finder Resource

The finder, with a principal method

```
FinderResults find(SearchRecipe sr)
```

is discussed in Chapter 3, "Resource Data, Searches & Vocabularies". It enables discovery of Resources matching the SearchRecipe, and optionally putting them in an order of preference, by examination of the Resource Descriptions in the Metadata.

Mailbox

E-speak has both Outboxes and Inboxes, but only Inboxes are exposed to Clients as Core-managed Resources. The Core's only actions on outboxes are the `startServingOutbox()` and `stopServingOutbox()` methods of the `CoreManagementInterface`. An Inbox is where a Client gets messages from the Core. A Client can have more than one Inbox, but each Inbox must be explicitly connected by the Client before it can be used to receive messages.

An Inbox cannot be exported.

The Inbox class implements the `MailboxInterface` defined below:

```
interface MailboxInterface
{
    boolean isConnected()
    throws ESInvocationException;

    void connect(int slot)
    throws ESInvocationException;

    void disconnect()
    throws ESInvocationException;

    void reconnect(int slot)
    throws ESInvocationException;
}
```

An Inbox is a Core-managed Resource that provides a unidirectional communication channel from the e-speak Core to a Client. When a Client registers a Resource with the e-speak Core, it must assign an Inbox Resource as the "Resource Handler" for the Resource. Any service requests directed to the Resource are delivered to the Client on the I/O channel associated with the Inbox that was named the Resource Handler.

An Inbox can be in one of the two states: connected or disconnected. Upon creation, the Inbox starts in the connected state. The creator of the Inbox becomes the owner of the Inbox, and the Inbox is set up to use the I/O channel information passed with the request to create the Inbox. The Inbox remains in the connected state until the Client requests an explicit disconnect, or until the I/O channel associated with the Inbox is closed, at which time it is put in the disconnected state. If a Client sends a message to a Resource whose handler is an Inbox in the disconnected state, an exception is thrown by the e-speak Core.

One may argue that Inboxes are unnecessary and that the e-speak Core could store the I/O channel information in the Resource Handler field directly. There are two main reasons for having the Inbox store the I/O channel information and not the Resource- one has to do with Client restart, and the other with delegation. These are explained in the following subsections

Inbox and Client Restart

In the e-speak environment, a Client can recover from some types of failures, one of which is the failure of a Client process. If a Client process dies and restarts, it can reconnect to the Core, discover and activate its previous Protection Domain, and discover and connect to the Inboxes owned by it. That way it can continue to serve the Resources that were registered by it during its previous incarnation.

Connecting to an Inbox involves updating the I/O channel information maintained by the Inbox. Keeping the I/O channel information in the Inbox helps simplify the Client's job at restart. It only has to discover and re-connect to one or a few inboxes. If, instead, the I/O channel information is stored in all the Resources registered by the Client, it would somehow need to be updated all over the place upon reconnection by the Client.

Name Frame

- 1 Client A passes the name of the Inbox IB to the other Client, along with a certificate to perform a reconnect operation on the Inbox.
- 2 Client B requests the e-speak Core to reconnect it to the Inbox IB. The Core replaces Client A's I/O channel information with Client B's I/O channel information.
- 3 Any further service requests directed to any of the 100 Resources are diverted to the I/O channel specified by Client B. The process of reconnection is performed atomically. Though logically the reconnect operation involves a disconnect operation on behalf of Client A and a connect operation by Client B, no one really sees the transient disconnected state.

Name Frame

A Name Frame manages the bindings of ESNames to Resources. A Client's default Name Frame is part of its Protection Domain. This section first describes the structure of an ESName and a binding and then describes Name Frames and data structures used by Name Frames.

ESNames

The only way a Client can refer to a Resource when communicating with the Core is to specify an ESName for the Resource. ESNames are defined fully in Chapter 6, “Communication”, Section “ESNames” on page 165”.

Bindings

In e-speak, a name is bound to a *Mapping Object*, which consists of an array of Accessors. An Accessor can be one of two types, as represented in Table 6.

Table 6 Mapping Object accessor types and descriptions

Accessor Type	Descriptions
Search request	A set of attributes, their corresponding values, and a Vocabulary to use in interpreting them
Explicit binding	A single instance of a Resource

Thus, a name can be bound to:

- Zero or more Resources
- Zero or more Search Recipes
- Some combination of explicit bindings and search request bindings

The term *simple binding* is applied to a name bound to a Mapping Object that has a single explicit binding. The term *complex binding* is used otherwise.

NameSearchPolicy

A NameSearchPolicy is used when a find request has returned some bindings. The NameFrameInterface listBindings or listNames methods listed below are used with a NameSearchPolicy argument:

```
class NameSearchPolicy
{
    static final int NSP_ANY = 0;
    static final int NSP_SIMPLE = 1;
    static final int NSP_EXPLICIT = 2;
    static final int NSP_PARTIAL = 3;
    ESName contract;
    int bindingType;
    boolean matchSense;
}
```

NSP_ANY means match any binding types. NSP_SIMPLE means match simple binding types. NSP_EXPLICIT means match explicit binding types. NSP_PARTIAL means match partial binding types (this is not implemented in the current release, and will cause undefined behavior if used).

If matchSense is false, the meaning of the Name Search Policy is negated, so listBindings will return the names of bindings that do not satisfy the Name Search Policy.

Name Frame Methods

Some NameFrame methods throw ESServiceException. Chapter 7, "Exceptions" lists the exception hierarchy for NameFrame methods.

The NameFrame methods are defined below:

```
interface NameFrameInterface
{
    boolean isBound(String baseName)
    throws ESInvocationException;

    void bind(String baseName, SearchRecipe recipe)
    throws NameCollisionException, QuotaExhaustedException,
    ESInvocationException, ESServiceException;

    void rebind(String baseName, SearchRecipe recipe)
    throws ESInvocationException, NameCollisionException;

    void unbind(String name)
    throws ESInvocationException, InvalidNameException,
    QuotaExhaustedException;

    void rename(String oldName, String newName)
    throws ESInvocationException, ESServiceException,
    InvalidNameException, NameCollisionException;

    void copy(String toName, ESName from)
    throws ESInvocationException, ESServiceException,
    NameCollisionException, InvalidNameException,
    StaleEntryAccessException, QuotaExhaustedException;

    void add(String name, ESName from)
    throws ESInvocationException, InvalidNameException,
    StaleEntryAccessException;
```

```

    void subtract(String name, ESName from)
    throws ESInvocationException InvalidNameException,
    StaleEntryAccessException;

    String[] listNames(NameSearchPolicy nsp)
    throws ESInvocationException, NameNotFoundException;

    String[] listBindings(String aBaseName,
    NameSearchPolicy nsp,
    ESName targetFrame)
    throws ESInvocationExceptionInvalidNameException,
    StaleEntryAccessException, QuotaExhaustedException;
}

```

A Name Frame can be exported by value or by reference. In the case of export by value, the Name Frame state is the bindings ESMaP. The serialization for ESMaP is defined by the e-speak serialization format. ESMaP is an ESArray in which the convention is that consecutive elements are treated as pairs. In the case of bindings, the first element of a pair is the string component of ESName; the second is a MappingObject to which ESName is bound. A MappingObject consists of a set of SearchRecipes and explicit bindings to resources. The explicit bindings are internal pointers (repository handles) to the resource metadata in the e-speak Core's repository. A MappingObject is serialized as an ESSet containing the SearchRecipes in the MappingObject (explicit bindings are not contained in the serialized form transmitted in the case of pass by value).

All methods that create a new entry in a Name Frame return a Name Collision Exception if the name already appears in the target Name Frame. An explicit rebind or unbind is required before the name can be reused.

The `isBound` method checks to see if the specified name (`baseName`) is bound in this Name Frame. It returns true if the name is bound.

The method `bind` binds SearchRecipe to a specified name (`baseName`) in this Name Frame.

The method `rebind` changes the binding of the specified name (`baseName`) in this Name Frame to the new SearchRecipe.

The method `unbind` removes the binding from NameFrame.

The method `rename` renames the binding associated with `oldname` to `newname`.

The method `copy` copies the binding of `from` to `toName`.

The method `add` adds the binding of `from` to the binding of `name` to give a new binding for `name`.

The method `subtract` subtracts the bindings of `from` from the bindings associated with `name` to give a new binding for `name`.

The method `listNames` returns an array of strings corresponding to all bindings that match `NameSearchPolicy nsp`. The Name Search Policy allows the Client to specify the type of binding and/or Contract in which the Resource is registered.

The method `listBindings` lists all the bindings of the argument `aBaseName` that match `NameSearchPolicy nsp`. These bindings are placed in the `NameFrame` named by `targetFrame`. The return value is an array of `String`, each element being the name of a new binding in `targetFrame`.

Protection Domain

A Client's Protection Domain is analogous to a user's home directory in an operating system. It contains a root Name Frame in which the Client can place bindings.

Each Protection Domain is associated with a quota. The goal of this is to track and manage use of space in the Repository. To support this, each Protection Domain has three fields associated with it: used, soft limit, and hard limit. A Protection Domain is guaranteed to be able to allocate Resources up to its soft limit. A Protection Domain may be able to allocate Resources up to its hard limit, depending on the memory usage of the Core. The default hard limit is 10,000,000 bytes, and the default soft limit is 30,000 bytes.

A Protection Domain cannot be exported.

The `ProtectionDomain` interface is defined below:

```
interface ProtectionDomainInterface
{
    ESName[] switchPD()
```



```
throws ESInvocationException, PermissionDeniedException,
NameNotFoundException StaleEntryAccessException,
QuotaExhaustedException;
```

```
Object[] getQuotaInfo()
throws ESInvocationException PermissionDeniedException,
NameNotFoundException;
```

```
Object[] setQuota(long softQuota, long hardQuota)
throws ESInvocationException, PermissionDeniedException,
NameNotFoundException;
```

```
ESName newProtectionDomain(String name,
boolean persistent
)
throws PermissionDeniedException;
}
```

The method `switchPD` switches the Client's active Protection Domain to this Protection Domain (i.e., the Protection Domain receiving the method invocation). It returns an array of two `ESNames`. Element [0] is the `ESName` for the old Protection Domain, and element [1] is the `ESName` for the new Protection Domain.

The `Object []` array returned by `getQuotaInfo` and `setQuota` contains at least three values. The first is `Long` containing the total number of bytes currently consumed in the Core by this Protection Domain. The second is `Long` containing the soft limit in bytes. The third is `Long` containing the hard limit in bytes for this Protection Domain.

The method `newProtectionDomain` creates a new Protection Domain. The `name` parameter is the name given when registering the new Protection Domain in the default vocabulary. The parameter `persistent` is set to true, if the new Protection Domain is to be made persistent. The return value is the `ESName` of the new Protection Domain.

The following initial names are defined in the default `NameFrame` of a new Protection Domain:

"CurrentPD" is bound to the Protection Domain itself

"Core" is bound to the core name frame (`es://host/core`) (see Chapter 6,

"Communication", section on `ESNames`).

Remote resource manager

The Remote Resource Manager is described in Chapter 6, "Communication".

Repository

This is described in Chapter 10, "Repository (Informational)".

Repository View

A Repository View contains references to a set of Resources.

When a Client does a find in a Repository View, the Core will attempt to match only those Resources included in the view. If no match is found, no accessor is added to the Mapping Object.

A Repository View can be exported by reference or by value.

The RepositoryView class is defined below:

```
class RepositoryView
{
    ESName[] Resources;

    boolean add (ESName res)
    throws ESInvocationException PermissionDeniedException,
    StaleEntryAccessException, NameNotFoundException,
    QuotaExhaustedException;

    boolean remove (ESName res)
    throws ESInvocationException PermissionDeniedException,
    StaleEntryAccessException, NameNotFoundException,
    QuotaExhaustedException;

    boolean contains (ESName res)
    throws ESInvocationException PermissionDeniedException,
    StaleEntryAccessException, NameNotFoundException;
```

```
    boolean clear ();  
    throws ESInvocationException QuotaExhaustedException  
    PermissionDeniedException, NameNotFoundException;  
  
    boolean addExternalLookupHandler(ESName res);  
    throws ESInvocationException PermissionDeniedException,  
    StaleEntryAccessException, NameNotFoundException;  
  
    boolean removeExternalLookupHandler()  
    throws ESInvocationException StaleEntryAccessException  
    PermissionDeniedException, NameNotFoundException;  
}
```

An externalLookupHandler is not used in this release. Any attempt to use addExternalLookupHandler or removeExternalLookupHandler will cause undefined behavior.

In general, all methods return true if they are successful, false if they fail. Clients can add Resources to and remove Resources from a Repository View. Attempts to add a Resource already in a Repository View will fail, as will attempting to remove a non-existing Resource. The method clear removes all Resources from the Repository View. The method contains returns true if the Resource, res, is contained in the Repository View.

Two Resource Contracts are available at system start-up in addition to those for Core-managed Resources. The default Resource Contract allows any Client to register a Resource. It is useful for Clients wishing to define Resources that don't specify a particular interface, such as Callback Resources. The second Resource Contract is for creating new Resource Contracts.

A Contract can be exported by value or by reference.

```
class ResourceContract
{
    ESName[] Vocabularies;
    string type;

    void getVocabularies(ESName targetFrame);
    throws ESInvocationException PermissionDeniedException,
    StaleEntryAccessException, NameNotFoundException;
}
```

The method `getVocabularies` populates the Name Frame, `targetFrame`, with the names of the Vocabularies supported by the Resource Contract. The Name Frame `targetFrame` is cleared before the operation.

Developer Release X.03.03.00, September 2000

Resource Factory

A Client wishing to register a Resource with an e-speak Core uses the Resource Factory. This is also used for creating Core-managed Resources.

The `ResourceFactoryInterface` class is defined below:

```
class ResourceFactoryInterface
{
    void registerResource (
        ResourceDescription descr,
        ResourceSpecification spec,
        Boolean persistence,
        Object param,
        ESName targetFrame,
        String toBaseName
    )
    throws ESInvocationException PermissionDeniedException,
        StaleEntryAccessException, NameNotFoundException,
        NameCollisionException;
}
```

The `registerResource` method takes a `ResourceDescription` and a `ResourceSpecification` as parameters. If `persistence` is true, the Core will preserve the metadata after the Client's connection is closed, and also the state, in the case of a core-managed Resource only. The state of an external Resource is never preserved after the Client's connection is closed. The `targetFrame` parameter is the `ESName` of a Name Frame in which the name for the new Resource will be put. The `toBaseName` parameter is the name of the new Resource in the Name Frame. The `Object param` is intended to hold Resource-specific information for creating Core-managed Resources, but is not currently used. It can be of any type supported in the e-speak serialization format.

Every instance of `e-speak` provides a `MetaResource` that provides access to metadata (Resource Descriptions and Resource Specifications). Once a Resource has been registered using a Resource Factory, the only way to access its metadata is through a message sent to the `MetaResource`, using the `ResourceManipulationInterface` defined below.

```

interface ResourceManipulationInterface
{
    void unregister (ESName resource)
    throws ESInvocationException;

    void setResourceOwner (ESName resource)
    throws ESInvocationException;

    ESName getResourceOwner(ESName resource)
    throw ESInvocationException;

    ESName getResourceProxy (ESName resource)
    throws ESInvocationException;

    void setResourceProxy (ESName resource,
    ESName resourceHandler)
    throws ESInvocationException;

    ESName getResourceContract (ESName resource)
    throws ESInvocationException;

    ADR getMetadataMask(ESName target)
    throws ESInvocationException;

    void setMetadataMask(ESName target, ADR mask)
    throws ESInvocationException;

    ADR getResourceMask(ESName target)
    throws ESInvocationException;

    void setResourceMask(ESName target, ADR mask)
    throws ESInvocationException;

    ADR getOwnerPublicKey(ESName target)
    throws ESInvocationException;
}

```

```
void setOwnerPublicKey(ESName target, ADR key)
throws ESInvocationException

ESMap getPublicRSD(ESName resource)
throws ESInvocationException;

void setPublicRSD(ESName resource,ESMap rsds)
throws ESInvocationException;

ESMap getPrivateRSD(ESName resource)
throws ESInvocationException;

void setPrivateRSD(ESName resource, ESMap rsds)
throws ESInvocationException;

ResourceDescription getResourceDescription(ESName target)
throws ESInvocationException;

void setResourceDescription(ESName resource,
ResourceDescription desc)
throws ESInvocationException;

int getEventControl (ESName resource)
throws ESInvocationException;

void setEventControl (int setting)
throws ESInvocationException;

boolean isPersistent (ESName target)
throws ESInvocationException;

boolean isTransient (ESName target)
throws ESInvocationException;

void setPersistent (ESName target)
throws ESInvocationException;

void setTransient (ESName target)
throws ESInvocationException;

ESUID getESUID(ESName target)
throws ESInvocationException;

ESName getUrl(ESName target)
throws ESInvocationException;
```

```

    long getQuota(ESName target)
    throws ESInvocationException;

    ResourceType getType(ESName target)
    throws ESInvocationException

    ADR getServiceID(ESName target)
    throws ESInvocationException

    void setServiceID(ESName target, ADR id)
    throws ESInvocationException
}

```

All methods can throw `PermissionDeniedException`, `StaleEntryAccessException` and `NameNotFoundException`

The convention for a Resource-specific data (RSD) array is that it consists of a sequence of pairs- the first element of each pair is a string used to tag the second element. (This is represented in an `ESMap`, for example in the return from `getPublicRSD`).

Most of the methods in a `MetaResource` are for setting or getting the fields of its Resource metadata. Some aspects of these methods warrant explanation:

The `unregister` method removes (unregisters) the Resource, `resource`, from the Repository. This removes `ResourceDescription` and `ResourceSpecification`; no more messages can be sent to the Resource after this operation.

The `setResourceOwner` method sets the owner of the Resource, `resource`, to the `ESName` of the calling Client's Protection Domain.

The `setResourceProxy` and `getResourceProxy` methods set and get the Resource Handler.

There is no method for setting the Resource Contract, because this cannot be changed once the Resource has been registered.

The method `getQuota()` returns the total charge in bytes to the owner's quota due to that Resource.

The methods `getMetadataMask` and `setMetadataMask` are used for getting and setting the operations for which security is disabled for a particular Resource's metadata: anybody can invoke the methods listed in this mask to manipulate the particular Resource's metadata. The methods `getResourceMask` and `setResourceMask` perform the analogous function for the operations supported by the Resource itself.

The user Interface

This is not implemented in the current release

```
interface UserInterface {
    public String getDescription()
        throws PermissionDeniedException, NameNotFoundException;

    public AttributePropertySet getProperties()
        throws PermissionDeniedException, NameNotFoundException;

    public void mutateProperties (AttributePropertySet props)
        throws PermissionDeniedException, NameNotFoundException;
}
```

Vocabulary

See Chapter 3, "Resource Data, Searches & Vocabularies".

Appendix: Method Names

In messages sent to Core-managed Resources (see Chapter 6, "Communication") the method is identified by a string. The following strings are used.

```
AccountManagerInterface
PF_REGISTERUSER
```

PF_UNREGISTERUSER
PF_AUTHENTICATEUSER
PF_GETUSERPROFILE
PF_SETUSERPROFILE
PF_GETALLUSERS
PF_ADDDESCRIPTION

ConnectionManagerInterface
OPENCONNECTION
GETCONNECTIONS
CLOSECONNECTION
CLOSECONNECTIONFROMREMOTE

CoreManagementInterface
PING
GETCLIENTCONNECTIONS
STOPSERVINGOUTBOX
STOPSERVINGINBOX
STARTSERVINGOUTBOX
STARTSERVINGINBOX
REMOVEPROTECTIONDOMAIN
DENYNEWCLIENTSESSIONS
ACCEPTNEWCLIENTSESSIONS
GETTOTALMEMORY
GETFREEMEMORY
START_JVM_GC
STOP_JVM_GC
SET_JVM_GC_INTERVAL
GET_JVM_GC_INTERVAL
IS_JVM_GC_RUNNING
START_SCAVENGER
STOP_SCAVENGER
GET_SCAVENGER_STATS
SET_NUM_STATS
FinderInterface
FIND
FINDNEXT

MailboxInterface
ISCONNECTED
CONNECT
DISCONNECT
RECONNECT

ManagedServiceIntf (implemented by Core management resource)
GETNAME

GETDESCRIPTION
GETOWNER
GETUPTIME
GETVERSION
GETERRORCONDITION
GETSTATICINFO
COLDRESET
WARMRESET
START
STOP
SHUTDOWN
REMOVE
GETSTATE
GETVARIABLEENTRIES
GETVARIABLENAMES
GETVARIABLEENTRY
SETVARIABLE
GETRESOURCEENTRIES
GETRESOURCENAMES
GETRESOURCEENTRY
SETRESOURCE

NameFrameInterface
ISBOUND
BIND
REBIND
UNBIND
RENAME
COPY
ADD
SUBTRACT
LISTNAMES
LISTBINDINGS
NEW_SUB_FRAME

ProtectionDomainInterface
SWITCHPD
GETDEFAULTFRAME
SETDEFAULTFRAME
GETQUOTAINFO
SETQUOTA
NEW_PROTECTION_DOMAIN

RemoteResourceManagerInterface
EXPORTRESOURCE
IMPORTRESOURCEFROMMSG

IMPORTRESOURCE
EXPORTRESOURCEFROMMSG
UNEXPORTRESOURCE
UNEXPORTRESOURCEFROMMSG
UPDATEEXPORTEDRESOURCE
UPDATEEXPORTEDRESOURCEFROMMSG
UPDATEIMPORTEDRESOURCE
UPDATEIMPORTEDRESOURCEFROMMSG
EXPORTONCONNECTING

RepositoryViewInterface
ADD
REMOVE
CONTAINS
CLEAR
ADD_ELOOKUP
REMOVE_ELOOKUP

ResourceContractInterface
REGISTERRESOURCE
GETVOCABULARIES

ResourceFactoryInterface
REGISTER_RESOURCE

ResourceManipulationInterface
UNREGISTER
GETESUID
SETRESOURCEOWNER
GETRESOURCEOWNER
GETRESOURCEPROXY
SETRESOURCEPROXY
GETRESOURCECONTRACT
GETPUBLICRSD
SETPUBLICRSD
GETPRIVATERSD
SETPRIVATERSD
GETRESOURCEDESCRIPTION
SETRESOURCEDESCRIPTION
GETEVENTCONTROL
SETEVENTCONTROL
ISEXPORTEDBYVALUE
SETEXPORTTYPE
GETQUOTA
GETMETADATAMASK
SETMETADATAMASK

GETRESOURCEMASK
SETRESOURCEMASK
GETOWNERPUBLICKEY
SETOWNERPUBLICKEY
GETSERVICEID
SETSERVICEID
ISPERSISTENT
ISTRANSIENT
SETTRANSIENT
SETPERSISTENT
GETURL
GETTYPE

UserInterface (not implemented in the current release)

GETDESCRIPTION
GETPROPERTIES
MUTATEPROPERTIES

VocabularyInterface

GETDESCRIPTION
GETPROPERTIES
MUTATEPROPERTIES

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

[illegible]

The basis of e-speak access control is a Public Key Infrastructure (PKI). In the remainder of this chapter we assume the reader is familiar with the principles of PKI, sometimes also known as Public Key Cryptography. There are many texts to which the reader can refer [see for example *Schneier, Pfleeger, Stallings*].

The means by which a private key is protected is implementation dependent: not part of the architecture. It is very important that the private key is held securely, so it is not unintentionally made available to others. In the default implementation the private key is encapsulated inside a Private Security Environment (PSE) object, described below.

To decide whether to honor an incoming request a service must decide if the accompanying certificate (or certificates) grant access rights for the request. Before that, it verifies that the sender of the request knows the private key corresponding to the public key in the certificate to which the access rights have been given (formally this is the Subject of the certificate). It does this by a cryptographic protocol described in Chapter 6, “Communication”.

Finally before honoring the request, the service must verify that it trusts whoever issued the certificate. It does this by verifying that the certificate has been signed by an entity that it trusts.

Comparison with X.509 Certificates

The most common use of certificates is in X.509 based infrastructures to link an entity's name to its public key (technically the X.509 Distinguished Name). This is how certificates are used in the web. A drawback is that, typically, having used the certificate to verify the name, a service needs to consult an authorization database to determine the access to be granted.

E-speak certificates are more general than this. They are signed (authenticated statements) linking a public key to a Name or a Tag. (Certificates linking a Name to another Name also exist, and are described below.) The word "tag" distinguishes the field concerned from an X.509 "attribute", whose function is broadly similar. A Tag typically states an access right. Thus to make an access control decision a service does the following:

- Examines the tag in the certificate to see if it grants access
- Checks the entity making the request knows the corresponding private key
- Verifies the certificate has been issued (is signed by) an entity it trusts

X.509 name certificates are issued by entities called Certificate Authorities. To avoid confusion with this, in e-speak we refer to entities issuing certificates as Issuers. E-speak Issuers can issue either Name or Attribute certificates.

Another feature in e-speak not found in X.509 is that it implements a split trust model. An entity does not have to trust all Issuers equally. It need not trust any given Issuer at all. Those it does trust, it may only trust to issue certificates granting access to a subset of its operations.

Conversely, issuing certificates in e-speak is not a reserved prerogative: anyone can do it. Whether or not the certificate will grant access to any Resource depends on whether the Resource Handler trusts the Issuer for the service in question. The list of which Issuers are trusted for what is called Trust Assumptions. This is discussed later in this chapter.

Derivation from SPKI

E-speak implements the Simple Public Key Infrastructure (or SPKI) [see *RFC 2692-2693*]. In addition to the properties already described, SPKI specifies a structure and set of operations on Tag and Name certificates. These are used to parse and process the certificates when making access control decisions. The processing and access control is discussed later in this chapter. Certain tags (e-speak tags) are defined that will be checked explicitly by the e-speak infrastructure before an access is authorized. However, applications can choose to use any syntactically valid SPKI tag. E-speak will check that certificates containing such tags are valid, but will not use them for an access control decision. The application will have to interpret these non e-speak tags when making access control decisions. Core managed Resources will ignore non e-speak tags.

Certificate Management

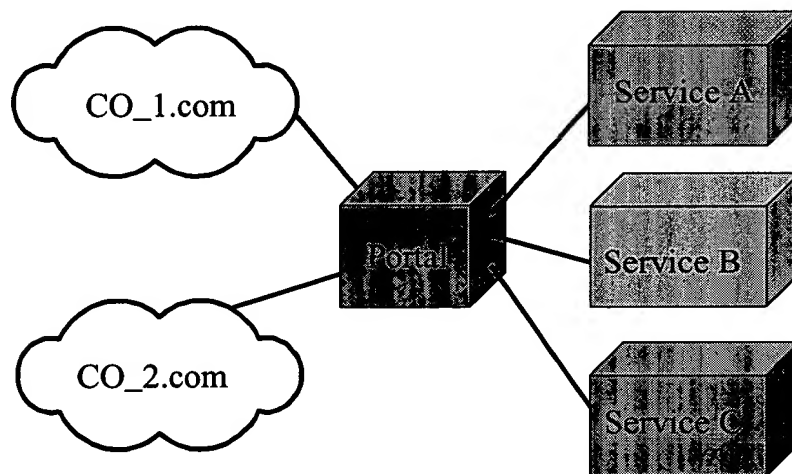
The process by which an Issuer decides to issue a certificate granting access rights to an entity is implementation dependent and therefore not part of the architecture. The general process would be for entities to register either with some Issuer or with a separate Registration Authority (RA). For registration the entity may need to provide credentials such as credit card number, social security number, bank details, employee ID, user id., and full name. Once the registering body is satisfied it will issue a certificate, or give instructions for issue. The registering body may be fully automated, or may queue registrations for human inspection.

A given entity may have several certificates that have been issued to it. If no strategy is adopted to structure and manage certificate issuing then there may be very large numbers of certificates required. Administrators and operators would find it difficult to run e-speak systems, and operations such as access revocation would be extremely hard. Hence we discuss and recommend certain strategies for certificate management. These are based around familiar concepts such as user-groups (or roles), found in several common operating systems. These are not part of the architecture. The management strategy practised must reflect the business requirements of the deploying organization.

Anybody can create a key-pair in e-speak and then register to get an Issuer to issue certificates to the public key. There is no notion of a centralized, all powerful, trusted Certificate Authority. Instead entities choose which Issuers they trust for what. Authentication in e-speak relies on proof of knowledge of the private key: there is no centralized authentication service. Hence the e-speak security architecture is a global, fully distributed and single sign-on.

Example of certificate-based Security (Informational)

Consider the diagram below. Two large ".com" companies are accessing a portal to use services provided by the portal. For simplicity we have shown only 3 services.



The data held by the services may be sensitive, so both companies would like to be sure that their employees are accessing the correct portal and services. In addition, having made arrangements for access to the portal (and paid fees), both companies might prefer to be responsible for managing their own lists of employees and control who can access the portal's services.

From the portal's point of view, it probably only wants to deliver services to paying customers and only to deliver those services that each customer has paid for.

Suppose CO_1 has done a deal with the portal to access services A, B and C, and CO_2 has done a deal to access service A and C only. Lets further suppose that CO_1 and CO_2 are each running an Issuer , called Issuer1 and Issuer2 respectively. The portal configures A and C to trust both Issuer1 and Issuer2; it configures B to trust Issuer1 only. Then CO_1 and CO_2 can issue certificates to each of their employees. CO_1's certificates will be honored at A, B and C, but CO_2 certificates will only be honored at A and C.

Each time a service sees a certificate from either company that grants access, it increments the bill for that company. This leaves each company in control of who among its employees gets access to the services for which it has paid. Each company is in control of revocation (e.g. if the employee leaves). In addition the portal can immediately revoke access to an entire company, by removing the company's Issuer from the list of trusted Issuers.

Each company may want to make sure that their employees are accessing only genuine services. To do so CO_1's Issuer issues a Tag certificate binding each of service A, B and C's public keys to a tag such as : "CO_1 approved". It must then ensure that its employees configure their clients to check for this tag before accessing the service. Similarly CO_2's Issuer issues a Tag certificate to services A and C conferring an attribute that is meaningful to CO_2.

Note that this requires very little authorization data to be held and managed by the portal. It only needs to remember the public keys of CO_1 and CO_2's Issuer. If access control were based on authenticating a name and mapping accesses to that name, then the portal would have to keep a list of all employees in each company that can access any of the services, and which accesses are allowed for each name - much more data to manage and maintain.

Authorization Data

The informal structure of an authorization certificate is:

Certificate header: a constant field starting " (cert "
 Issuer: the public key of the Issuer
 Subject: the public key or the name of the entity granted the certificate
 An optional "delegation" field
 Tag: Details of what is authorized
 Optional validity qualification and comment.

In this structure, it is the tag that requires most attention by client applications.

Tags

As e-speak implements SPKI, any valid SPKI tag can appear in a certificate. The BNF for SPKI is given in the *SPKI BNF Format* section. In this section we give some example SPKI tags that can appear in certificates and explain the BNF for a tag.

E-speak defines a set of standard tabs (see "E-speak Authorization Tags" on page 87), that will be checked automatically by the infrastructure. The examples given in this section are not standard e-speak tags, so they would have to be checked explicitly by the application.

An SPKI tag is an S-expression, that is a list enclosed in matching "(" and ")".

The BNF for a tag is:

```
<tag> = "(" "tag" <tag-expr>* ")" ;
<tag-and> = "(" "*" "and" <tag-expr>+ ")" ;
<tag-expr> = <byte-string> | <tag-simple>
            | <tag-prefix> | <tag-range>
            | <tag-set> | <tag-and>
            | <tag-star> ;
<tag-simple> = "(" <byte-string> <tag-expr>* ")" ;
<tag-prefix> = "(" "*" "prefix" <byte-string> ")" ;
<tag-range> = "(" "*" "range" <range-ordering> <low-lim>? <up-
lim>? ")" ;
<tag-set> = "(" "*" "set" <tag-expr>* ")" ;
<tag-star> = "(" "*" ")" ;
```

```

<tag-and> = "(" "*" "and" <tag-expr>+ ")" ;1
<range-ordering>= "alpha" | "numeric" | "time" | "binary" |
"date" ;
<up-lim> = <lte> <byte-string> ;
<low-lim> = <gte> <byte-string> ;
<lte> = "l" | "le" ;
<gte> = "g" | "ge" ;

```

A tag is a list of lists, with each list denoted by brackets. In its simplest form (tag-simple), a tag is simply composed of byte-strings. The access control machinery must interpret the meaning of the tag when making an access control decision. The following examples are adapted from SPKI examples previously published as Internet drafts. An example form for tags applying to a file system is:

```
(tag (files <pathname> <access> ))
```

An instance of such a tag is:

```
(tag (files //ftp.espeak.net/pub/EspeakArch.pdf read))
```

A client presenting a certificate containing the above tag is allowed read access to EspeakArch.pdf (assuming authentication was successful).

<tag-set> field

Groups of permissions can be granted using the "tag-set" form:

```
(tag (files //ftp.espeak.net/pub/EspeakArch.pdf (* set read
write)))
```

This grants read and write access to the file.

<tag-prefix> field

A set of permissions having a common prefix can be granted using the "tag-prefix" form:

```
(tag (files (* prefix //ftp.espeak.net/pub/ ) (* set read write)))
```

This grants read and write access to any file under the pub directory.

<tag-star> field

The "tag-star" form stands for the set of all valid s-expressions and byte strings.

¹ The <tag-and> field is an e-speak specific extension to SPKI.

The above tag grants all permissions on all files under pub.

Note that trailing "("" can be omitted. So the above is equivalent to:

The two last tags both grant all permissions on all files anywhere.

The above grants all permissions on anything. This might look as though it is conferring a lot of power. However, e-speak has a split trust model: the issuer of the certificate containing this tag might only be trusted by a single Resource.

The "tag-and" form is not used in writing a certificate. It expresses the authorizations conferred by the set of tags in the following expression. This is analagous to a set-intersection operation: the authorization resulting from a "tag-and" form will be that satisfying each and every one of the following tags. So it is more restrictive than that of any of the tags on its own. This form is used internally when authorization depends on more than one certificate. The process is described under *Tag Intersection*.

The "tag-range" form stands for the set of all byte strings lexically (or numerically) between the two limits. The ordering parameter (alpha, numeric, time, binary, date) specifies the kind of strings allowed. For example, the following tag indicates the authorization to issue purchase orders whose value is less than \$5000.

The following indicates a salary between \$50,000 and \$100,000

86

E-speak tags that authorize access to services have the following form:

The following tag authorizes the "stop" operation in the serviceManagementInterface for the identified Resource.

The forms `tag-star`, `tag-prefix`, `tag-set` and `tag-range` can all be used within an e-speak tag. So the following tag authorizes operations on the `ServiceManagement` interface in two different Resources.

The long strings at the end represent the ServiceId, described below.

```
(tag (net.espeak.method ServiceManagementInterface (*) (*) ))
```

```
(tag (net.espeak.method ServiceManagementInterface ))
```

```
(tag (
  (net.espeak.method
    (* set
      ResourceFactoryInterface
```

)

```
(tag (net.espeak.method file read (* prefix es.espeak.net/pub/
)))
```

Currently we have only defined e-speak tags for the Network Object Model. This assumes a set of services with one or more interfaces, each interface containing one or more methods. The programming of J-ESI and the interaction with core-managed Resources follow this model. However, e-speak can support other programming models: an XML document exchange model and a direct messaging model have both been implemented. The tags used by these models are part of the programming models. There are not part of the core architecture, since the core does not need to interpret them: the resource handlers do it.

The `serviceId` field in the Resource specification (see Chapter 3, "Resource Data, Searches & Vocabularies") can contain any valid SPKI tag-expression, defined as a "tag-expr" in the BNF (see "SPKI BNF Formats" on page 109). This tag-expression can be set by anybody with a certificate, from an Issuer trusted by the MetaResource, authorizing `setServiceId` in the MetaResource. The `serviceId` field is delivered to the resource handler with each message for the Resource.

Developer Release X.03.03.00, September 2000

```
<serviceId> = "(" "net.espeak.service" <service class> <service  
name> <unique id> ")"
```

- 1.) The name attribute in the Resource specification contract, if any.
- 2.) The contract type, if any.
- 3.) A 64-bit random no. if neither of the above exists.

- 1.) The "name" in the Resource description
- 2.) A 64-bit random no., if 1) is not found.

A secure random number generator should be used, so that the probability of accidental authorization when the default has been used will be infinitesimal..

The `serviceId` is intended for use by applications to identify services without using the Resource name or access path (ESNames). This decouples authorization from resource naming and has several advantages:

- Service ESNames can be changed without affecting authorization
- Authorization can be revoked by changing a service's identity, without changing its ESName
- In a replicated service replicas can all have the same identity
- Tag patterns (the "tag-star" form) can be used effectively, limiting the number of certificates issued

Developer Release X.03.03.00, September 2000

Protection of ServiceIds

Service identity plays a crucial role in authorizing access to a service (see "Verifying tags and tag intersection" on page 100). It is essential that the `setServiceID` operation source is protected, so that a valid certificate is required to invoke it.

Names: Userids, Groups....

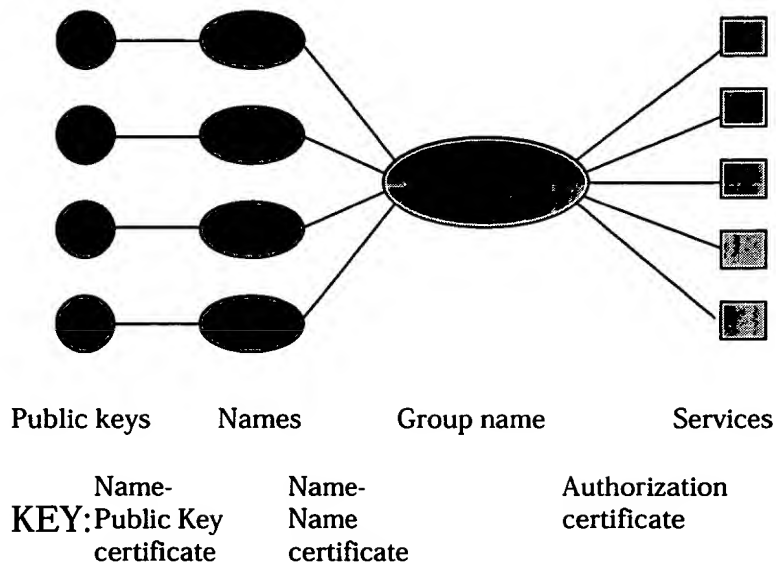
E-speak also supports SPKI name certificates, of two types. In the first place, an Issuer can issue a certificate that binds a public key to a name. This has similarities to X.509 certificates which bind a public key to an X.509 Distinguished Name. (A Distinguished Name is a name in a special format, distinguishing it globally from any other name.)

SPKI name certificates do not restrict the syntax of the name, other than requiring them to be a bytestring. Instead, names are scoped by the public key of the issuer. Referring back to our example (see "Example of certificate-based Security (Informational)" on page 82), both CO_1 and CO_2 could have an employee named John Doe. Assuming each company had an Issuer that issued name certificates binding these names to public keys, the fully qualified name for each John Doe is:

```
Public key of CO_1 issuer: John Doe
Public key of CO_2 issuer: John Doe
```

Hence the portal (and anybody else) would have no difficulty distinguishing between the two instances of John Doe.

The second type of name certificate binds a name to a name. For example we might want to bind John Doe to the name "users". This kind of certificate confers membership of the group "users" on the userid John Doe. It can be used to build a role- or group- based security model, such as represented below (see "Managing certificates (informational)" on page 106).



The algorithm which relates a public key to an authorization in this case is described below (see "Name Reduction" on page 97). [See also *RFC 2693*]

Certificate Structure

The two kinds of certificates in e-speak are Authorization Certificates that bind a tag to a public key or a name and Name Certificates that bind a name to a public key or a name. The following sections describe and explain the BNF which specify these types.

Some general features of the specification are:

Nearly every field begins with its name as a literal string.

* is used to mean "0 or more cases of the preceding field"

* is also used to mean "anything valid" in the tag-star field described above

"uris" means a field with one or more URI's.

The full SPKI BNF is given at the end of this Chapter (see "SPKI BNF Formats" on page 109).

30262

```
<cert> = "(" "cert" <version>? <cert-display>?
<issuer> <issuer-info>?
<subject> <subject-info>?
<deleg>?
<tag>
<valid>?
<comment>? ")" ;
```

Issuer field

```
<issuer> = "(" "issuer" <principal> ")" ;
<principal> = <public-key> | <hash-of-key> ;
<hash-of-key> = <hash> ;
<hash> = "(" "hash" <hash-alg-name> <hash-value><uris?> ")" ;
<hash-alg-name> = "md5" | "sha1" | <uri> ;
<hash-value> = <byte-string> ;
<public-key> = "(" "public-key" <pub-sig-alg-id> <s-expr>*
<uris?> ")" ;
<pub-sig-alg-id>= "rsa-pkcs1-md5" | "rsa-pkcs1-sha1" | "rsa-
pkcs1" | "dsa-sha1" | <uri> ;
```

92

the delegation field present. (It is the literal "propagate".) Suppose a service trusts the first Issuer directly, and not the second Issuer. If a client presents a certificate issued from the second Issuer, the service will need to see the delegate certificate conferring the privilege on the second Issuer before it authorizes access. The URIs would specify the location of delegate certificates. This is not used in the current version of e-speak. Instead, the required supporting certificates are obtained during the Session Layer handshake (see Chapter 6, "Communication"). The parser will ignore this field.

The "hash-alg-name" and "pub-sig-alg-id" fields identify algorithms used for hashing and for signature verification - usually the literal abbreviated algorithm names given. The "uri" alternative in each case could be used to give a URI of some other algorithm.

Subject field

The <subject> field denotes the entity to which the certificate is issued.

```
<subject> = "(" "subject" <subj-obj> ")" ;
<subj-obj> = <principal> | <name> | <object-hash> ;2
<principal> = <public-key> | <hash-of-key> ;
<name> = <relative-name> | <full-name> ;
<relative-name> = "(" "name" <byte-string>* ")" ;
<full-name> = "(" "name" <principal> <byte-string>* ")" ;
```

The <subject> is either a public key, a name or the hash of an object. If the subject is a public key, then the entity presenting the certificate must prove possession of the corresponding private key before authorization is granted. This uses the cryptographic protocols described in Chapter 6, "Communication".

If the <subject> is a name, then authorization is granted to the entity that has a certificate binding that name to its public key (see "Name Certificates" on page 96). Several certificates may be required to prove this. For example the authorization

² The definition here departs from SPKI slightly. SPKI defines <subj-obj> = <principal> | <name> | <object-hash> | <keyholder>; E-speak does not support <keyholder>. If the parser encounters a keyholder field, it will throw an exception. Which exception depends on the point from which it is invoked. One of the e-speak exceptions specified in Chapter 7, "Exceptions" will be thrown.

```
<subject-info> = "(" "subject-info" <uris> ")" ;
```

The intent of this field is to provide a list of URIs that provide information about the subject. For example if the subject is a hash of a key, it might provide the location of the key being hashed. If the subject is a name, it might provide the location of the name certificates. This field is not used in the current version of e-speak. The parser will ignore it.

Delegation field

The optional <deleg> field is defined as follows.

```
<deleg> = "(" "propagate" ")" ;
```

If this field is included in a certificate, then the subject is authorized to delegate the authorization specified in the certificates tag. The subject does this by issuing certificates containing the tag, or a subset of the tag's privileges. This is discussed further under *Delegation*.

Validity field

The optional <valid> field is defined as follows.

```
<valid> = <valid-basic> <online-test>* <restrictions> ;
<valid-basic> = <not-before>? <not-after>? ;
<not-after> = "(" "not-after" <date> ")" ;
<not-before> = "(" "not-before" <date> ")" ;
<date> = <byte-string> ;
```

If the valid field is missing, the certificate is assumed to be valid without constraints. The fields <online-test> and <restrictions> defined in [Working Draft] are not supported in the current version of e-speak; the parser will ignore them. The <valid-basic> field is used to support time-based revocation, as described under *Certificate Revocation*.

A <date> field is an ASCII byte string of the form:

```
YYYY-MM-DD_HH:mm:ss
```

This is always UTC. For example, "1997-07-26_23:15:10" is a valid date. So is "2001-01-01_00:00:00". "MM" is a two digit integer in the range 1 to 12; "mm" and "SS" are two-digit integers in the range 0 to 59.

The optional comment field is defined as follows:

```
<comment> = "(" "comment" <byte-string> ")" ;
```

Anything in this field is intended to provide information to humans. It is ignored by e-speak.

Name Certificates

The format for name certificates is:

```
<name-cert> = "(" "cert" <version>? <cert-display>?
               <issuer-name> <issuer-info>?
               <subject> <subject-info>?
               <valid> <comment>? ")" ;
<issuer-name> = "(" "issuer" "(" "name" <principal> <byte-
string> ")" ")" ;
<principal> = <public-key> | <hash-of-key> ;
```

The characteristic feature of a name certificate is the the <issuer-name> field. This defines the issuer of the certificate plus the name of the certificate holder. "Issuer-name" does not mean "name of issuer". The byte-string in this field is the certificate holder's name, and the <principal> is the issuer's public key, or a hash of it. In the latter case, there may be a following field containing a URI of the full key, but this is not currently used or supported in e-speak.

Public Keys

An example of a public key is:

```
(public-key
  (rsa-pkcs1-md5
    (e #03#)
    (n
      |ANHCG85jXFGmicr3MGPj53FYYSY1aWAue6PKnpFErHhKMJa4HrK4WSKTO
      YTTlapRznnELD2D7lWd3Q8PD0lyi1NJpNzMkxQVHrrAnIQoczeOZuiz/yY
      VDzJ1DdiImixyb/Jyme3D0UiUXhd6VGAz0x0cgrKefKnmjy410Kro3uW1|
    ))
)
```

The long string between "|" 's is a number encoded in base64 notation for relative brevity. This is a feature of BNF advanced syntax [see *BNF Notation* below].

Such items may have to be written in certificates, but in the following text, we use "PK XXX" as an abbreviation for "XXX's public key".

Example

Taking the example (see "Names: Userids, Groups...." on page 90), the following certificate could be issued by CO_1's Issuer.

```
(cert      Certificate A
  (issuer (name (PK CO 1) "John Doe"))
  (subject (PK John Doe))
  (not-after "2001-01-01_00:00:00")
)
```

The underlining is referred to in the next paragraph.

Name Reduction

The objective of name reduction is to reduce the name that appears in a subject field to a single public key, a <principal>. Name reduction replaces the name in a subject field, by rewriting it with the subject field from the corresponding name certificate. It uses the fact that a fully qualified name in a subject field has the same format as <principal> <byte-string> in an issuer-name field. For example, given **Certificate A** above, suppose there is an authorization certificate:

```
(cert
Certificate B
  (issuer PK X)
  (subject (name (PK CO 1) "John Doe"))
  (tag (net.espeak.method CoreManagementInterface ))
)
```

The two underlined fields being the same, we can replace <subject> in B by <subject> in A, giving certificate C:

```
(cert
Certificate C
  (issuer PK X)
  (subject (PK John Doe))
  (tag (net.espeak.method CoreManagementInterface ))
)
```


Suppose an entity "Editor" issues a name certificate to "Foreign Desk"; and this entity in turn issues one to "Paris Correspondent". Each will have PK holder as its Subject. An authorization certificate could be made out as follows:

The subject field is a Compound Name. Accessing the name certificates implied in the subject field from left to right, we replace this field successively by:

- yielding a certificate which can be authenticated.

Wire format for certificates

Delegation

E-speak supports SPKI delegation. If an Issuer is not trusted directly by the entity checking the authorization, its certificates cannot effectively authorize more than the delegate certificate authorizes. The SPKI certificate reduction rules [see

Developer Release X.03.03.00, September 2000

Consider the following certificate.

Suppose Y now issues a certificate to Z as follows.

Here Y is attempting to authorize more, for longer than was contained in the certificate issued to it by X.

```
(cert
  (issuer PK X)
  (subject PK Z)
  (tag (net.espeak.method CoreManagementInterface ))
  (not-after "2000-10-01_00:00:00")
)
```

Developer Release X.03.03.00, September 2000

Verifying tags and tag intersection

Tag verification is the process of determining whether the set of certificates presented contain the required authorization. SPKI tags define sets of authorizations. For example the following tag authorizes all methods on all instances of the CoreManagementInterface.

```
(tag (net.espeak.method CoreManagementInterface ))
```

So the above tag "contains" the following tag (xxxxyyyyzzzz is the serviceID).

```
(tag (net.espeak.method CoreManagementInterface ping
xxxxyyyyzzzz))
```

Appending elements to the end of a tag reduces the set of authorizations specified. So:

```
(tag (net.espeak.method CoreManagementInterface ping))
```

specifies less than

```
(tag (net.espeak.method CoreManagementInterface ))
```

In the case of a delegation chain, where the successive certificates authorize:

- 1 services A, B, C
- 2 services B, C, D
- 3 services B, D, E -

the only service authorized will be B - the only member of the "intersection" of the three certificates.

Implementing Verification

In e-speak, each time an object receives a request to invoke a method, the security infrastructure will check that there is a certificate that contains the tag needed to invoke the operation. For Core-managed Resources the security infrastructure is contained in the core. For other Resources, it is part of the Resource. The security infrastructure is part of the current implementation of J-ESI, and clients can use the security infrastructure API's for their own resource handlers.

For this to work the infrastructure must know the serviceID of the Resource. The serviceID is part of the Resource's metadata, and the core presents the serviceID with each request. It is trusted to present the correct serviceID.

Authorization certificates can be issued to names as well as public keys. If a certificate issued to a name is presented that authorizes the operation, the name must be reduced to the public key of the invoker, as described in the *Name Reduction* section. The invoker's public key will be authenticated by the protocols described in Chapter 6, "Communication".

In addition to the e-speak tags specified in the *E-speak Authorization Tags* section, a client or service can ask for application-specific tags to be checked, by invoking the security infrastructure APIs. Since no e-speak tags are specified for servers to present to clients, any authentication of the service by the client will be application-specific. For example a client might check for a tag identifying the Id. of a service, such as:

This means that the server will have to get a certificate issued to it containing this tag. See the *Certificate Issuers and Registration* section below.

The security APIs for checking application-specific tags are outside the architecture. They are application-specific, and no application-specific tags are supported for core-managed Resources.

[illegible]

```
(net.espeak.method net.espeak.examples.ExampleIntf foo)
```

This tag masks method foo in interface net.espeak.examples.ExampleIntf and method bar in interface net.espeak.examples.Example2Intf

```
(net.espeak.method (*set
(net.espeak.examples.ExampleIntf foo)
(net.espeak.examples.Example2Intf bar)
```

This tag masks all methods beginning with foo:

```
(net.espeak.method net.espeak.examples.ExampleIntf (* prefix
foo))
```

This tag masks methods foo and bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf (* set foo
bar))
```

To mask methods with prefix foo or bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf
(* set (* prefix foo) (* prefix bar)))
```

To mask all methods in the interface:

```
(net.espeak.method net.espeak.examples.ExampleIntf )
```

This is equivalent to

```
(net.espeak.method net.espeak.examples.ExampleIntf (*))
```

since missing trailing elements match anything.

To mask methods foo in InterfaceA and bar in InterfaceB:

```
(* set (net.espeak.method InterfaceA foo)
(net.espeak.method InterfaceB bar))
```

To mask all methods:

```
(net.espeak.method)
```

or simply

```
(*)
```

008002T" 2203E260

0000089206

Note that sometimes we may want to have the same identity for multiple services. For example, the services might be replicated. So, whether service identities are required to be unique and how this is enforced is not part of the architecture.

Trust Assumptions (Informational)

The basis for establishing trust assumptions is:

- Who you trust and for what.
- The importance of protecting this information from tampering.
- The need to conceal or to reveal who you trust.

All this is application specific, and trust assumptions are not part of the e-speak core's architecture.

Trust assumptions define whose certificates will be honored, and the acceptable set of tags in each case. Both clients and services may have trust assumptions. Trust assumptions do not appear in any of the e-speak protocols (core to core, or client to core APIs).

It may be important for a client or server not to reveal certain trust assumptions, containing information of potential use to an attacker. Conversely, a trust assumption might need to be broadcast, for example to let potential (paying) clients know the Issuer they need to get a certificate from, to access a service.

It is essential to prevent unauthorized tampering with trust assumptions, so that attackers cannot add themselves to the list of trusted entities.

The current implementation uses self-issued certificates to store trust assumptions. A certificate is only accepted as a trust assumption if it is self-issued. The format of trust assumption certificates in the current implementation of e-speak is exactly like that of an authorization certificate. The client or service must distinguish between authorization certificates and trust-assumption certificates. This should be easy: authorization certificates will be exchanged between two entities as part of the message protocols (see Chapter 6, "Communication"). Trust assumption certificates will probably be stored locally on disk. They should in any case be separate from authorization certificates.

The following certificate authorizes the entity CertificateIssuer to issue certificates authorizing any method in the CoreManagementInterface.

```
(cert
  (issuer PK self)
  (subject PK CertificateIssuer)
```



```
(tag (net.espeak.method CoreManagementInterface ))
)
```

The following certificate means the entity trust itself to issue any certificate.

```
(cert
  (issuer PK self)
  (subject PK self)
  (tag (* ))
)
```

Note that trust assumptions can use names or public keys as subjects.

Certificate Revocation

In the current implementation the only supported means of expressing validity is time (the <valid-basic> element). Once a certificate is issued it is valid until it expires.

SPKI supports online tests for validity. Future releases of e-speak will probably do the same, and support the principle of a certificate revocation list (CRL).

Managing certificates (informational)

The current implementation of e-speak has a Certificate Issuing Service (CI) that can be used to issue certificates authorizing access to services that trust it. This CI might be used to manage access to a set of services on a set of e-speak cores. Here we outline the way in which the CI manages its certificates as a guideline to those who may wish to implement their own CI.

The CI implements a notion of users and groups. When a user registers with the CI, this service issues a name certificate binding the user's name (userid) to a public key. Thereafter all certificates are issued to the userid rather than the user's public key. This means that to revoke all access to a user we need only revoke the certificate binding the userid to the public key.

The CI provides a certificate directory interface from which stored certificates can be retrieved. This allows services to see what certificates have been issued to users and permits users to retrieve certificates that have been issued to them.

Note that all a user's power is revoked as soon as the certificate binding their name to a public key expires.

Renewing keys

The CI supports key renewal by issuing a certificate binding the user's new key to the user's name. All other certificates issued to the user will remain valid as they are issued to the user's userid (name). The user may have to undergo a process similar to registration to convince the CI that the new key is valid.

Private Security Environments (Informational)

Private keys must never be shared and so need to be stored securely. How private keys are stored is a matter for the owner of the key and has no impact on the e-speak protocols or APIs used to interact with the core. It is therefore not part of the architecture. In the current implementation a PSE or Private Security Environment is used. This stores the keys in an encrypted file on a disk.

The private keys are never revealed to the the application. Instead data is sent to the PSE object when it requires signing. The PSE framework has been designed so that the underlying mechanisms can be changed to accomodate devices like smart cards.

Interoperability with X.509 (Informational)

X.509 certificate infrastructures [see *RFC 2459*] are becoming more and more common. An X.509 certificate binds an entity's distinguished name to its public key. This is very similar to the way in which a SPKI name certificate binds a name to a public key. One difficulty is that in SPKI the Issuer is denoted by a public key. In X.509 the Certificate Authority is denoted by a "distinguished name". Its public key is not required to be in the certificate. When it isn't there, the Certificate Authority is assumed to have a well-known public key.

An e-speak CI can take an X.509 certificate, verify it (check it is signed by a trusted Certificate Authority) and issue an e-speak Name Certificate binding an encoding of the subject's X.509 distinguished name to the subject's public key. (This is not supported in the current release.)

In addition the e-speak certificate verifier could be extended to handle X.509 name certificates natively, automatically converting them to SPKI name certificates as outlined above. (This is not supported in the current release.)

X.509 version 3 also supports attribute certificates and work is on going within the IETF on defining a profile for attributes to use within the Internet's PKIX infrastructure. It is not possible to define a useable mapping from X.509 attribute certificates into SPKI authorization certificates, as X.509 attributes can be arbitrary. In principle it should be possible to define a mapping from SPKI certificates into X.509 attribute certificates.

SPKI BNF Formats

E-speak uses two BNF syntaxes. The "advanced" syntax is used for manually-input data and human reading. It has been used throughout this document. Its advantages for this purpose are allowing white spaces (including line-feed), and base64 and hex codings of numbers. The base64 coding allows public keys to be written with relative brevity.

The parser accepts certificates in advanced syntax or canonical syntax, and outputs them in canonical syntax. This is used for all internal operations, such as protocol exchanges, and for serialized transmission. All hashes are computed on data in canonical syntax. This is necessary, because varying numbers of white spaces would produce invalid hashes.

Advanced Syntax

The advanced syntax follows. Its initial non-terminal is <s-part>.

```
<alpha> = [a-zA-Z];
<base64> = "##" (<base64-char> | <space>)* "##" ;
<base64-char> = <alpha> | <digit> | [+/=];
```

```

<bytes> = <token> | <string> | <raw-bytes> | <quoted-string> |
<base64> | <hex> ;
<byte-string> = <display-type>? <bytes> <decimal> = [0-9]+ ;
<digit> = [0-9];
<display-type> = "[" <bytes> "]" ;
<hex> = "#|" (<hex-digit> | <space>)* "|";
<hex-digit> = [0-9A-Fa-f];
<punctuation> = [\-. / _ : * + =] | [' ` , @] | [% ^ & \ [ \ ] # ~ < > ? : |] ;
<quoted-string> = "#<" {delimiter char c} {delimiter string s
not containing c} {c}
                    {any character strng not containing s} {s} ;
<raw-bytes> = "#" <decimal> "*" {binary byte string of that
length} ;
<s-expr> = "(" (<s-part> | <space>)* ")" ;
<space> = [ \t\r\n]*;
<s-part> = <byte-string> | <s-expr> ;
<string> = "\" {string chars} "\"";
<token> = (<alpha> | <punctuation>)
(<alpha> | <punctuation> | <digit>)* ;

```

We also allow end-of-line comments indicated by !. Comments are treated as white space.

Within a string C conventions may be used, including octal escape sequences. Specifically:

```

\b backspace (010)
\f formfeed (014)
\n newline (016)
\r return (015)
\t tab (011)
\nnn octal escape

```

Where nnn is a 3-digit octal numeric in the range 0₈ - 177₈, which is 0₁₀ - 127₁₀

Canonical Syntax

The canonical syntax defines the following.

```

<bytes> = <raw-bytes> ;
<decimal> = [1-9] [0-9]* | "0" ;
<s-expr> = "(" <s-part>* ")" ;

```

This disallows space in lists, all byte forms except counted string, and insists that decimal numbers have no redundant leading zeros. Hashes are always computed over canonical forms.

Within certificates, lists must start with a byte string and be non-empty:

```
<s-expr> = "(" <bytes> <s-part>* ")" ;
```

The following is the BNF currently recognized. The top-level non-terminals are:

```
- <cert>: a certificate.
- <name-cert>: a name certificate
- <proof> : a certificate justification.
  <proof> is used in the messaging protocol. (See ]
```

In the messaging protocol (See Chapter 6, "Communication") we use tag lists for queries and requirements.

```
<cert> = "(" "cert" <version>? <cert-display>?
        <issuer> <issuer-info>?
        <subject> <subject-info>?
        <deleg>? <tag> <valid>? <comment>? ")" ;
<cert-display> = "(" "display" <byte-string> ")" ;
<comment> = "(" "comment" <byte-string> ")" ;
<date> = <byte-string> ;
<date-expr> = <byte-string> ;
<deleg> = "(" "propagate" ")" ;
<full-name> = "(" "name" <principal> <byte-string>+ ")" ;
<gte> = "g" | "ge" ;
<hash> = "(" "hash" <hash-alg-name> <hash-value><uris>? ")" ;
<hash-alg-name> = "md5" | "sha1" | <uri> ;
<hash-of-key> = <hash> ;
<hash-value> = <byte-string> ;
<issuer> = "(" "issuer" <principal> ")" ;
<issuer-info> = "(" "issuer-info" <uris> ")" ;
<issuer-name> = "(" "issuer" "(" "name" <principal> <byte-
string> ")" ")" ;
<low-lim> = <gte> <byte-string> ;
<lte> = "l" | "le" ;
<name> = <relative-name> | <full-name> ;
<name-cert> = "(" "cert" <version>? <cert-display>?
        <issuer-name> <issuer-info>?
        <subject> <subject-info>?
        <valid> <comment>? ")" ;
<not-after> = "(" "not-after" <date> ")" ;
<not-before> = "(" "not-before" <date> ")" ;
<n-val> = <byte-string> ;
<object-hash> = "(" "object-hash" <hash> ")" ;
<one-valid> = "(" "one-time" <byte-string> ")" ;
<online-test> = "(" "online" <online-type> <uris>? <principal>
<s-part>* ")" ;
```

```

<online-type> = "crl" | "reval" | "one-time" ;
<principal> = <public-key> | <hash-of-key> ;
<proof> = (<cert> | <name-cert>)*
<public-key> = "(" "public-key" <pub-sig-alg-id> <s-expr>*
<uris>? ")" ;
<pub-sig-alg-id>= "rsa-pkcs1-md5" | "rsa-pkcs1-sha1" | "rsa-
pkcs1" | "dsa-sha1" | <uri> ;
<range-ordering>= "alpha" | "numeric" | "time" | "binary" |
"date" ;
<relative-name> = "(" "name" <byte-string>+ ")" ;
<requires> = "(" "requires" <tag>* ")" ;
<restrict-date> = "(" "date" <date-expr> ")" ;
<restriction> = <restrict-date> | <target> | <requires> ;
<restrictions> = <restriction>* ;
<reval> = "(" "reval" <version>? <subj-hash> <reval-body> ")" ;
<reval-body> = <one-valid> | <valid-basic> ;
<signature> = "(" "signature" <hash> <principal> <sig-val> ")" ;
<sig-val> = <s-part> ;
<subject> = "(" "subject" <subj-obj> ")" ;
<subject-info> = "(" "subject-info" <uris> ")" ;
<subj-hash> = "(" "cert" <hash> ")" ;
<subj-obj> = <principal> | <name> | <object-hash>;
<tag> = "(" "tag" <tag-expr>* ")" ;
<tag-and> = "(" "*" "and" <tag-expr>+ ")" ;
<tag-expr> = <byte-string> | <tag-simple>
| <tag-prefix> | <tag-range>
| <tag-set> | <tag-and>
| <tag-star> ;
<tag-simple> = "(" <byte-string> <tag-expr>* ")" ;
<tag-prefix> = "(" "*" "prefix" <byte-string> ")" ;
<tag-range> = "(" "*" "range" <range-ordering>
<low-lim>? <up-lim>? ")" ;
<tag-set> = "(" "*" "set" <tag-expr>* ")" ;
<tag-star> = "(" "*" ")" ;
<target> = "(" "target" <tag-expr>* ")" ;
<up-lim> = <lte> <byte-string> ;
<uri> = <byte-string> ;
<uris> = "(" "uri" <uri>* ")" ;
<valid> = <valid-basic> <online-test>* <restrictions> ;
<valid-basic> = <not-before>? <not-after>? ;
<version> = "(" "version" <byte-string> ")" ;

```

The elements <reval>, <online-test> (and related elements such as crl) and <restrictions> are parsed but silently ignored in the current implementation. Architectural extensions will be introduced to support these elements.

- Developer Release X.03.03.00, September 2000**

00802F" 220E260

[illegible]

Overview

The only way for a Client to request access to a Resource from a Resource Handler is to send a message through the e-speak core. The only way for a Resource Handler to return a reply to a Client is to send a message through the e-speak core. Thus, the core mediates all access between Clients and Resource Handlers. It is the only entity to accept connections: Clients and Resource Handlers establish connections to an e-speak Core so that they can communicate with each other.

All messages handled by e-speak cores are Protocol Data Units (PDU's), described below.

The possible exchanges of messages follow the Session Layer Security protocol “Session Layer Security Protocol (SLS)” on page 132.

Mediation by the e-speak core may include:

- Determining to which Inbox to route the message.
- Determining how to route the message (its routing path).
- Processing and transforming the message headers and contents. Only limited processing of the message is possible if security is enabled, implying that a Message Authentication Code (MAC) is appended. If so, fields of the PDU header which have been used to form the MAC must not be changed “Authentication of messages” on page 144.

Mediation is transparent to the Client and Resource Handler.

The e-speak core keeps no state information about messages beyond the time needed to complete processing. It does not keep any information about replies to messages. As far as the core is concerned, a reply is another message. It doesn't distinguish between clients and resource handlers: an entity which sends it a message is a "client" so far as it is concerned, and the destination is simply an Inbox.

If a Client needs a reply, it may wait or send another message; all messaging is asynchronous. Each asynchronous message has an identifier set by the sender. A reply can refer to this identifier so the Client knows to which message the reply has been sent.

Figure 6 shows the flow of messages through an e-speak core when a Client sends and receives messages from a Resource Handler.

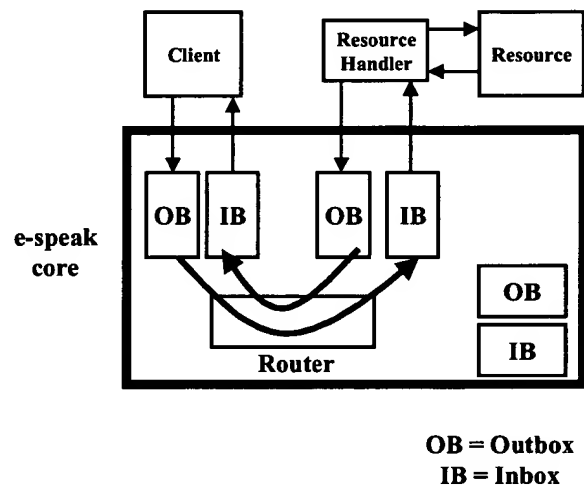


Figure 6 Message flow with an e-speak core

A client is not restricted to resources connected to the same e-speak core as itself. Figure 7 shows the message flow when a Client sends and receives messages from a Resource Handler on a remote e-speak core. The same protocol is used to exchange messages between e-speak cores as is used to exchange messages between Clients and e-speak Cores.

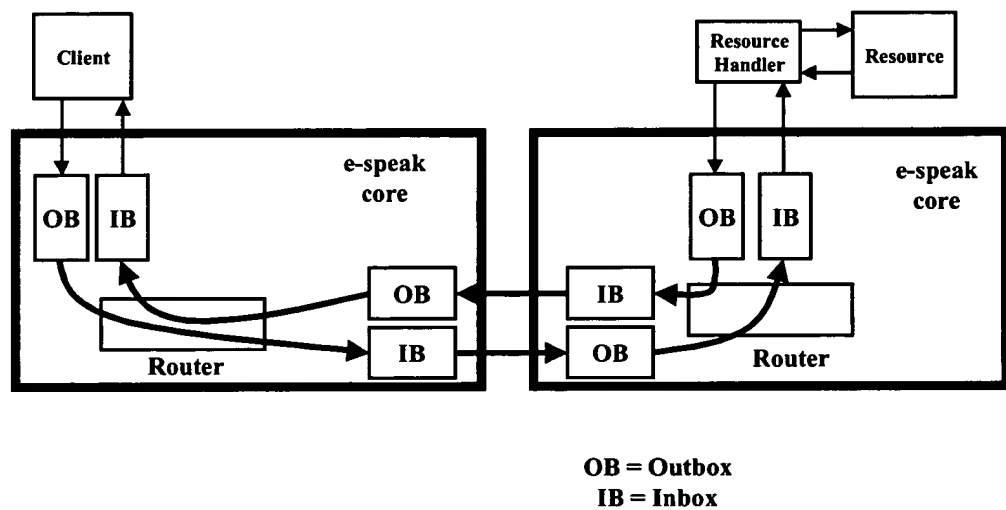


Figure 7 Core-to-core message flow

The creation and management of core-to-core communications is described below (see "Core to core communication" on page 151).

Protocol Data Unit (PDU)

All messages exchanged between e-speak cores, and between e-speak cores and clients are PDUs. A single PDU corresponds to a single Session Layer Security (SLS) Message.

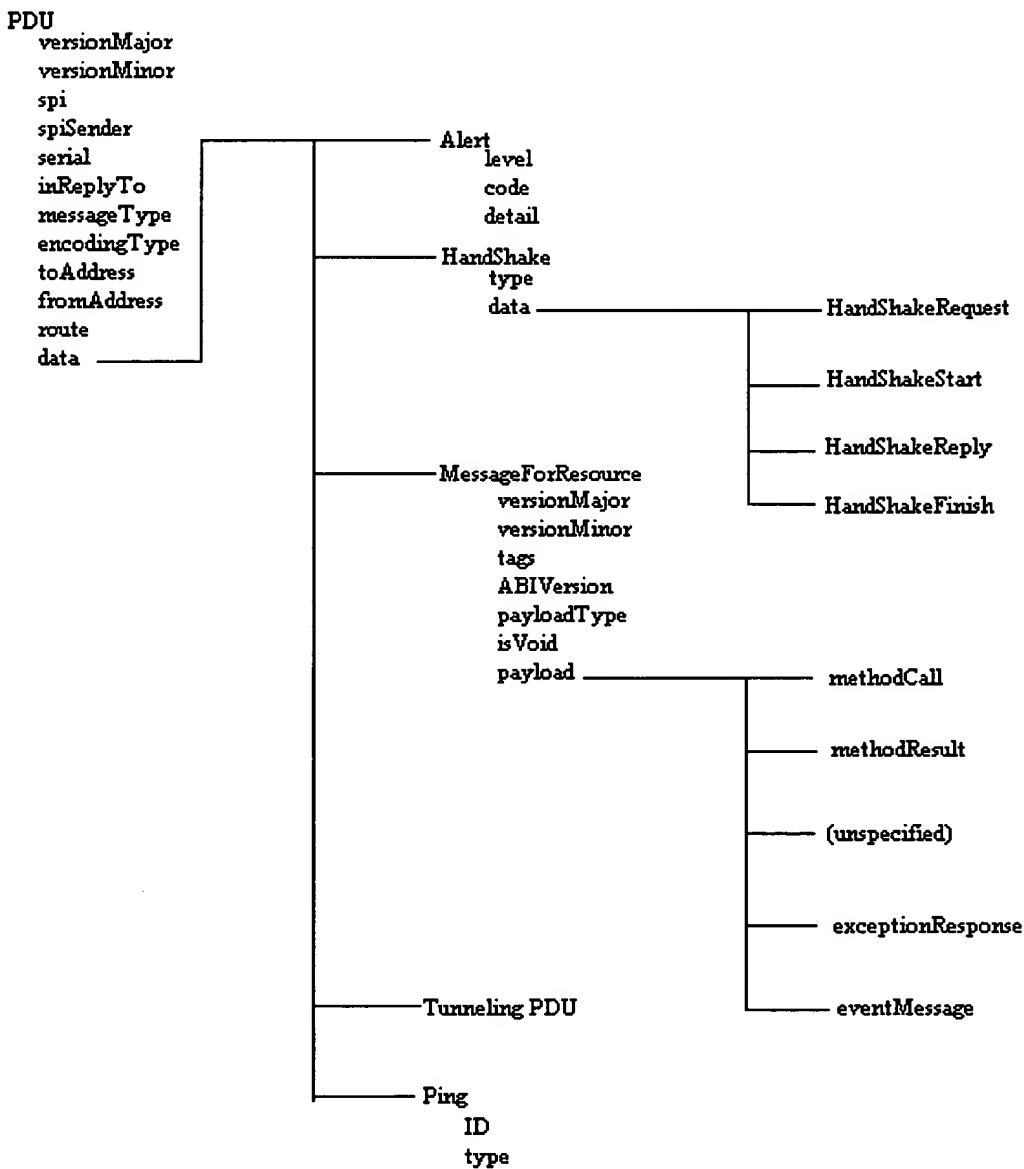
A class hierarchy for PDU's is shown in Figure 8. The members PDU.data, HandShake.data and MessageForResource.payload are all byte-arrays which in principle could hold an object of any class. Which class-instances are present in a PDU is given by the values of:

- PDU.messageType - for the contents of PDU.data
- HandShake.type - for the contents of HandShake.data, if present.
- MessageForResource.payloadType - for the contents of MessageForResource.payload, if present

The permissible values of PDU.messageType and the corresponding contents of PDU.data are:

ALERT (0)	Alert
HANDSHAKE (1)	HandShake
APPLICATION_MESSAGE (2)	MessageForResource
TUNNEL (3)	(Tunneling) PDU
PING (4)	Ping

Figure 8 PDU class hierarchy



The class PDU is:

```
class PDU {  
    int versionMajor;  
    int versionMinor;  
    int spi;  
    int spiSender;  
    int serial;  
    int inReplyTo;  
    int messageType;  
    int encodingType;  
    String toAddress;  
    String fromAddress;  
    byte[] route;  
    byte[] data;  
}
```

Version no.s

The current value for versionMajor is 1, and for versionMinor is 0.

SPI members

SPI stands for Session Parameter Index. This is used by the two endpoints in the Session Layer Security protocol to indicate which session the message is being sent on. Since the sender and the receiver may identify the SPI separately we have two fields: spi denotes the recipient's SPI; spiSender denotes the sender's SPI.

Message serial no. fields

Two fields are used by SLS to protect against replay attacks: serial is set by the sender; inReply to is the serial field of the message to which the sender is responding.

Message types

The following values for message type are defined: alert(0), handshake(1), application message(2), tunnel(3), ping(4). Alert, handshake, tunnel and ping are used in SLS to manage sessions "Session Layer Security Protocol (SLS)" on page 132. These types are described in the following section.

Encoding types

The following encoding types are defined for a PDU: clear data (0); protected data(1); secure data(2). Protected data is authenticated and protected from tampering by a Message Authentication Code (MAC) [see *Schneier pp. 455-459*]. Secure data is protected by a MAC and also encrypted for confidentiality.

Addresses

The String toAddress is an absolute ESName [see *ESNames* section below] and denotes the destination for the message. The String fromAddress is also an absolute ESName. It denotes the sender of the message and can be used for replies. The e-speak core attempts to resolve these names in its Root name-frame by finding the Mapping Object (see Chapter 4, "Core-Managed Resources") associated with each ESName. The Mapping Object is used by the e-speak core to refer to a Resource, a Search Recipe, or any combination.

If the e-speak core cannot unambiguously identify the Resource Handler for the toAddress, it will send an exception message to the Client. The format for an exception message is described in Chapter 7, "Exceptions". The possible exceptions are shown in Table 7 on page 131.

Route member

The byte-array route can be used by applications to pass routing data. This is never encrypted or protected by a MAC.

Data member

The format of this byte-array is determined by the encoding type. If the encoding is clear data, the byte array is a message body, of contents indicated by the message type.

If the encoding is secure data, then it has been encrypted according to the cipher negotiated in the SLS session set-up. Once it has been decrypted, it will have the same format as protected data: a MAC, followed by the message body. The contents of the message body is indicated by the message type. It will be an object of one of the classes described below.

These elements of a PDU are marshalled in the order of member definition shown in the class declaration above.

Marshalling of PDU Elements

In general, PDU's are marshalled according to the PDU marshalling format (PDFM) described below (see "PDU Marshalling format" on page 170). This applies to the PDU class listed above. However, for application messages the e-speak serialization format (ESF) (see "E-speak Serialization Format" on page 170) is used in principle. Currently this is not compatible with SPKI certificate formats (see Chapter 5, "Access Control"), so some messages are serialized partly using PDFM and partly using ESF. Where this happens we label each field either (*PDFM*) or (*ESF*). Otherwise we label the class corresponding to the message type.

Top Level PDU Message types

The data member of a PDU will contain an instance of one of the following classes, depending on PDU.messageType, as explained above. The elements of each message are marshalled in the order in which they appear in the class definition.

Alert

```
Class Alert{ (PDFM)
  byte level;
  byte code;
  String detail;
}
```

The Alert message is used for SLS session management. Valid level values are:

```
fatal (0x00)
warning (0x01)
debug (0x02).
```

All codes are normally sent with a level of fatal, unless indicated. Valid codes are:

- CLOSE_NOTIFY (0x00) (warning)
- UNEXPECTED_MESSAGE (0x01)
- BAD_SPI (0x0A)
- BAD_SERIAL (0x0B)
- BAD_MAC (0x0C)

- The `detail` String is intended for human consumption and is left unspecified.

These are newly introduced, to handle the following situations:

- ## Handshake

Developer Release X.03.03.00, September 2000

Application message

```
class MessageForResource { (PDFM)
byte versionMajor;
byte versionMinor;
ADRLList tags;
short secondaryABIVersion;
byte payloadType;
boolean isVoid;
byte payload[]; (ESF this field only)
}
```

ADRList is a list of SPKI tags using the *-set form as defined in “SPKI BNF Formats” (Chapter 5, “Access Control”)

isVoid indicates whether or not there is a payload. If isVoid is true, there is no payload and it is not marshalled or unmarshalled.

Tunnel

If the message type of PDU is TUNNEL, the data field of the PDU contains another PDU. The outer PDU is removed. The PDU contained in the data field is unmarshalled and forwarded to the address contained in the **toAddress** field of the inner PDU. The contents of the inner PDU, except for the **toAddress** field, may be encrypted - the object is to pass encrypted messages across a firewall.

```
class ping{ (PDFM)
String ID;
byte type;
}
```

The current value of ID is "SLS:Ping:v1.0".

- Request (0x00)
- Reply (0x01)

The value of HandShake.type indicates the contents of HandShake.data, as follows:

<u>TYPE VALUE</u>	<u>CLASS-INSTANCE IN DATA</u>
HANDSHAKE_REQUEST (0)	HandShakeRequest
HANDSHAKE_START (1)	HandShakeStart
HANDSHAKE_REPLY (2)	HandShakeReply
HANDSHAKE_FINISH (3)	HandShakeFinish

- The current value of the **ID** member is "SLS:HandshakeStart:v1.0".
- The current values of **majorVersion** and **minorVersion**, if present, are 0x01 and 0x00 respectively. They indicate the highest version of SLS supported by the sender.

```
class HandShakeRequest { (PDFM)
String ID;
boolean flag;
PDU pdu;
}
```

The boolean flag is set to true if the request includes a PDU, otherwise no PDU is included. The PDU is intended to be used for synchronization: it would contain the last message between the two parties. Currently it is not used.

```
class HandShakeStart { (PDFM)
String ID;
byte majorVersion;
byte minorVersion;
int spi;
ADR group;
ADR keyData;
ADR cipherSuiteList;
ADR tags;
ADR query;
}
```

spi is the session parameter index of the sender of this message.

keyData is the sender's part of the Diffie-Hellman key-exchange.

tags is the list of SPKI tags the sender is requiring the receiver to prove.

Handshake Reply

signature is the signature of the hash of this message, the previous HandshakeReply and the HandshakeRequest message.

053021 1000

```
methodCall
methodResult
exceptionResponse
eventMessage
(unspecified)
```

```
interfaceName = "core"
methodName = "bootstrap"
arguments = null
```

Routing to External Resources

A message to an external resource is also sent as a PDU to the e-speak core, but with the payloadType OBJECT and a payload unspecified by e-speak. The e-speak core will route this message to the resource handler if it can.

It cannot do so if the "To" field of the message is not a valid Resource, or if the Inbox specified in the destination Resource metadata is not connected to a Resource Handler, or if the Resource Handler's Inbox is full.

When it cannot deliver the message, the e-speak core will return an error (exception) message to the Client, if it can. If the Client's Inbox can't take the error message for any reason, the e-speak core discards the message.

Normally, the e-speak core places the following data in the route field of the PDU:

```
class routeData{
  String slot; (PDFM)
  boolean specificationNonNull; (PDFM)
  ESmap privateRSD; (ESF)
  ADR mask; (PDFM)
  ADR serviceID; (PDFM)
}
```

The slot field is used to enable many Inboxes to share a single channel (TCP connection in the current implementation). The slot identifies which Inbox the message is for.

If specificationNonNull is set to false, the three fields following are not marshalled.

These three fields are parts of the resource's metadata held by the e-speak core.

The privateRSD field is the resource's private RSD.

The mask field tells the resource handler which methods have security disabled.

The serviceID field is the service identity for the resource. Both these fields are <tag-expr> as defined in Chapter 5, "Access Control" - "SPKI BNF Formats".

Payload of messages from Core-Managed Resources

Initial Connection to the Core

The e-speak Core listens on a TCP port for Client connections (the default port is 12345). When it receives a connection request (see "Initial Connection Request" on page 128) a TCP channel is created between the Client and the Core. The Core creates a default protection domain for the Client and sends a PDU back to the Client, of type `MessageForResource`, with `payloadType` set to `METHOD_RESULT`. The `methodResult` in the payload will contain a `bootstrapReply` object:

```
class bootstrapReply{ (ESF)
    ESname Inbox;
    String InboxSlot;
    ESname CallbackResource;
    ESname ExceptionHandlerResource;
    String anchor;
}
```

`Inbox` and `InboxSlot` are the `ESnames` of the inbox and the slot allocated by the e-speak core to the client. The `CallbackResource` field is the `ESname` to be used to send messages to the client. This should be used in the `fromAddress` field of PDU's sent by the client.

The `ExceptionHandlerResource` is deprecated and should not be used.

The `anchor` field is the URL of the root name-frame of the Protection Domain which has been created by the e-speak core for the client.

Normal reply

A PDU sent from a Core-managed Resource, in reply to a `methodCall` when no exception has been thrown, has the payload type `METHOD_RESULT`, and the payload member contains a `methodResult` object:

```
public class methodResult
{
    Ob result; (ESF)
}
```

The `bootstrapReply` sent in response to a connection request is a special case of `methodResult`.

Exceptions

When an exception is thrown the payloadType is EXCEPTION and the payload member of messageForResource is an instance of exceptionResponse:

```
public class exceptionResponse {  
    Ob result; (ESF)  
}
```

One of the following exceptions will be thrown when the e-speak core cannot unambiguously identify the Resource Handler for a given toaddress:

Table 7 Exceptions for unresolved Resource Handler

Exception	Description
NameNotFoundException	The lookup procedure failed to find a Mapping Object.
UnresolvedBindingException	The only accessors in the Mapping Object are Search Recipes.
MultipleResolvedBindingException	The explicit bindings in the accessors refer to Resources with different Resource Handlers.
UndeliverableRequestException	The Resource Handler does not have the Resources needed to receive this message, or the Handler Inbox is not currently connected.

Events

When the core generates an event it will send a PDU with the data field containing a messageForResource, which will have the payloadType set to EVENT. The payload will be an eventMessage.

```
public class eventMessage  
{  
    Ob result; (ESF)
```

Messages from a Resource Handler to a Client

E-Speak implements a peer-to-peer communications model for messaging.

The Core does not distinguish between a message sent from a Client to a Resource Handler and a reply from the Resource Handler back to the Client. The Resource Handler sending a reply to a Callback Resource is treated as the Client, and the Client receiving the reply is treated as the Resource Handler for the Callback Resource.

Clients may have more than one Inbox. The only way for a Client to receive a message from any other Client is to register a Resource listing one of its Inboxes in the Resource Handler field of the metadata. Clients can manage different classes of messages by registering different Resources designating different Inboxes. Clients can also deal with different message classes by associating certain classes with Events.

Session Layer Security Protocol (SLS)

The Session Layer Security (SLS) Protocol determines all possible exchanges of PDU messages between clients, e-speak cores and resource handlers. The protocol, used between a client and a resource handler, may establish a secure session between them. It can also be used to establish an open, non-secure session. All SLS communication is carried in PDU messages, and all PDU messages are sent under the SLS protocol.

A secure session has the following properties.

- All messages exchanged between the two end points are authenticated. This prevents messages being changed or messages being inserted into the TCP connection by a third party (e.g. an attacker).
- All message exchanged between the two end points are protected against replay. This prevents a third party capturing the messaging and replaying it at a later date to trigger a repeat of the action taken by the recipient.

In a non-secure session messages are exchanged without encryption, authentication or protection against replay.

- Transport independence: SSL links a security session with a TCP socket. If the socket dies the security session dies with it: something undesirable when the life expectation of a security session is very different from the life expectation of the transport. Also, we cannot multiplex several security sessions onto the same socket, or perform dynamic load balancing of the end-point without starting the session from scratch. Moreover, even though properties like reliability and in-order delivery of messages are critical for a security protocol, some TCP details are not, and this might put unnecessary restrictions on the applicability of SSL. Finally, in some cases we might want to use a different transport for sending and receiving messages (i.e., outgoing messages use a different firewall needing two sockets). SLS tries to make the minimum number of assumptions on the communication transport solving most of the issues above.
- Tunnelling support: During firewall traversal we might want the firewall to control the client access rights to the internal LAN for every packet. However, we might not want the firewall to see all the traffic in clear (therefore, losing the end-to-end security property). This is difficult to achieve with SSL because either we let the client open a direct socket to the service or the firewall will see all the traffic in clear. On the other hand, with SLS we can nest a secure session inside another one, possibly with different end points, allowing to achieve both goals simultaneously.
- Elliptic cryptography: Most implementations of SSL only support Diffie-Hellman key agreement algorithms based on exponentiation. SLS uses a faster algorithm based on Elliptic Curve Cryptography (ECC) described by [Seroussi and Smart].

- Attribute certificates using SPKI [see *RFC 2692 - 2693*]. SSL only supports X509 name certificates, mainly to authenticate that the end-point "owns", according to a configured "trusted CA", the web address that we wanted to reach. Only one certificate by each party can be used, and in most cases only server authentication is performed. On the other hand, SLS performs a negotiation of tags that need to be proven represented by multiple SPKI certificates. This allows a fine grained control of security by mapping tags to actual permissions, raising the level of abstraction from "a stream of bytes" in SSL to a particular operation on service X in SLS, making it easier to integrate with application level security. Details on the use of SPKI certificates in SLS can be found in Chapter 5, "Access Control".
- Latency minimization: SLS allows the client to send application data after a round-trip negotiation has succeeded. In SSL two round-trips are needed before the application data is sent. This can have important performance implications when network delays are large and we need a quick response from the server.

Functional Description of SLS

In this section we describe the expected behavior of the protocol.

Protocol message types

Every SLS message is embedded in a PDU (Protocol Data Unit) (see "Protocol Data Unit (PDU)" on page 118), which contains header information allowing the system to dispatch it to the correct security context, route it through the network, identify replies, ensure the protocol version and so on. Two fields of this header relevant to our discussion classify messages according to the type of encoding and their purpose:

Supported encoding types

- CLEAR_DATA: The message is not encrypted or protected against modification.
- PROTECTED_DATA: The message is not encrypted but it is protected against modification with a signed digest (MAC)
- SECURE_DATA: The message is encrypted and protected against modification using a MAC.

Supported message types

- **HANDSHAKE:** Message exchanged during the key-agreement protocol. There are four types of handshake messages that will be discussed later.
- **ALERT:** Message that identifies an abnormal situation during the handshake or after the session has been established. ALERT messages can be fatal, forcing session termination, or just warnings, for which the response is implementation dependent.
- **APPLICATION_MESSAGE:** Message that communicates application data.
- **TUNNEL:** A message that contains another PDU in its payload. This is used to nest sessions, something important for firewall traversal.
- **PING:** A heartbeat message that is used by the session scavenger to know if the session is still active.
- **REKEY:** Forces a key offset of the instantiated cipher suite based on the previously negotiated shared secret. (REKEY is not supported in the current implementation.)

High level protocol state machine

Figure 9 High level state transitions in SLS.

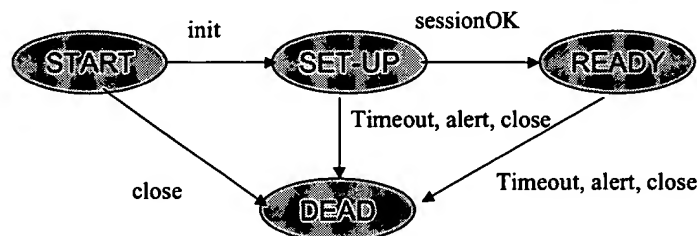


Figure 9 shows the possible states of a session, and what triggers transitions between them. There are four possible states:

- **START:** the session object has been created but it is not fully configured. Also, the key agreement protocol has not started.

- 05200001

- **init:** complete the initialization of the session object and send a message to the other party (if needed) to start the key agreement.
- **sessionOK:** The key agreement has finished successfully and we have a new cipher suite to install in the session.
- **timeout:** a timeout expired. The different timeouts are explained in detail below (see "Timeouts description" on page 146).
- **alert:** an authenticated (or optionally non-authenticated) fatal alert was handled/sent forcing a shutdown of the session.
- **close:** a client forced termination of a session by invoking the `close()` method.

SLS Handling a PDU

Depending on the state of the session, and the encoding/type of the PDU, the session behaves differently while handling PDUs. Table 8 on page 138 shows the expected behavior of a session when handling a PDU.

136

- MAC: either PROTECTED_DATA or SECURE_DATA encoding type. Obviously, the important property is that the PDU is correctly authenticated, otherwise we will always ignore the message regardless of its claims.
- Exception: notify the client doing the handling that the session is not operational.
- Ignore: do not take any significant action based on that PDU (i.e., change session internal state). Optionally, an implementation could log the event that a PDU is being ignored.
- Optional: an action is considered optional if an implementation can decide to ignore the PDU instead.
- Warning: send a warning ALERT response to the other party.
- HandleHsh: handle a HANDSHAKE PDU by making progress in the key agreement protocol. This could involve an Internal Send of a HANDSHAKE or ALERT PDU to the other party.
- HandleAl: handle an ALERT PDU. This might involve closing the session if it is a fatal alert, or logging the event otherwise
- HandleApp: handle an APPLICATION_MESSAGE PDU. Typically, the PDU will be passed in clear text to the client if it authenticates and/or decrypts correctly; otherwise it is ignored.
- HandleTun: handle a TUNNEL PDU. This could involve "peeling off" the outer PDU, returning the inner one (after decryption/authentication of the outer one) or calling a custom handler to deal with it.
- HandlePin: handle a PING PDU. This handling might require sending a reply PING PDU or just record that our previous PING has been replied successfully.
- HandleRe: handle a REKEY PDU. Forces the re-key of the handler part of the crypto suite.
-

Table 8 PDU handling behaviour depending on state

TYPE	ENCODE	START	SET_UP	READY	DEAD
HAND-SHAKE	CLEAR	Exception	HandleHsk	Ignore	Exception
	MAC	Exception	Ignore	Warning	Exception
ALERT	CLEAR	Exception	HandleAl Optional	Ignore	Exception
	MAC	Exception	Ignore	HandleAl	Exception
APPLICA-TION	CLEAR	Exception	Ignore	Ignore	Exception
	MAC	Exception	Ignore	HandleApp	Exception
TUNNEL	CLEAR	Exception	Ignore	Ignore	Exception
	MAC	Exception	Ignore	HandleTun	Exception
PING	CLEAR	Exception	Ignore	Ignore	Exception
	MAC	Exception	Ignore	HandlePin	Exception
REKEY	CLEAR	Exception	Ignore	Ignore	Exception
	MAC	Exception	Ignore	HandleRe	Exception

SLS Sending a PDU

The sending operation will first invoke the required security processing (i.e., encoding, MAC computation) and then it will use the underlying transport to deliver the message at the other end. Note that handling a PDU might have as a side effect that another PDU is sent to the other party, i.e., a response to a handshake message during the key agreement. We call that case an internal send as opposed to an external send directly invoked by the client.

Table 9 shows the expected behavior when trying to send a PDU through a session.

Table 9 PDU sending behavior depending on state

TYPE	MODE	START	SET_UP	READY	DEAD
HAND-SHAKE	Internal	NotApply	OK	OK	NotApply
	External	Exception	Retry	OK	Exception
ALERT	Internal	NotApply	OK	OK	NotApply
	External	Exception	Retry	OK	Exception
APPLICATION	Internal	NotApply	NotApply	NotApply	NotApply
	External	Exception	Retry	OK	Exception
TUNNEL	Internal	NotApply	NotApply	NotApply	NotApply
	External	Exception	Retry	OK	Exception
PING	Internal	NotApply	NotApply	OK	NotApply
	External	Exception	Retry	OK	Exception
REKEY	Internal	NotApply	NotApply	OK	NotApply
	External	Exception	Retry	OK	Exception

Some terms in that table deserve further explanation:

- **NotApply:** it is an implementation error if the protocol tries to send this message. This is only relevant for internal messages: the implementation does not have control over possible external messages.
- **OK:** this means that the session will perform the appropriate processing and try to deliver it to the lower layer. This does not mean that the message has been correctly sent, because this depends on the status of the underlying transport/session.

- **Retry:** The client is informed that the session is currently unavailable to send messages but this might change in the future.
- **Exception:** The client is informed that the session is permanently unavailable.

The key-exchange protocol

The key-exchange protocol is an authenticated Diffie-Hellman key exchange. From the session key agreed in the Diffie-Hellman exchange further keys are derived for encryption and confidentiality.

Features of the exchange are:

- **Elliptic Diffie-Hellman key exchange** instead of modulus exponentiation. Instead of choosing a group and checking its validity at the other end, we pick one of a pre-determined family of elliptic curves [see *Seroussi & Smart*]. Each party uses a random point on the curve as a private key, generates a second point from the first to send as a public key, and checks that the point he receives belongs to the same curve.
- There is no current support for multiple public keys of the same principal. This extension should be trivial by adding more than one signature in the handshake.
- **Added tunneling support:** we allow the responder to notify in its first handshake message that it wants to relay the session. Tunneling is described below (see "Support for tunneling" on page 149). When tunneling is indicated, the responder might not have to prove the tags requested.
- **Randomized Session Parameter Indices (SPIs).** We want SPIs to be hard to guess, to avert denial-of-service attacks. If SPI's are predictable, it is too easy to flood the client/server with fatal alert messages. At the same time, we want to be able to pay attention to non-authenticated alerts during the handshake (handling alerts is described in (see "Handling alert messages" on page 145)).

003021" 22055760

Figure 10 Key agreement protocol

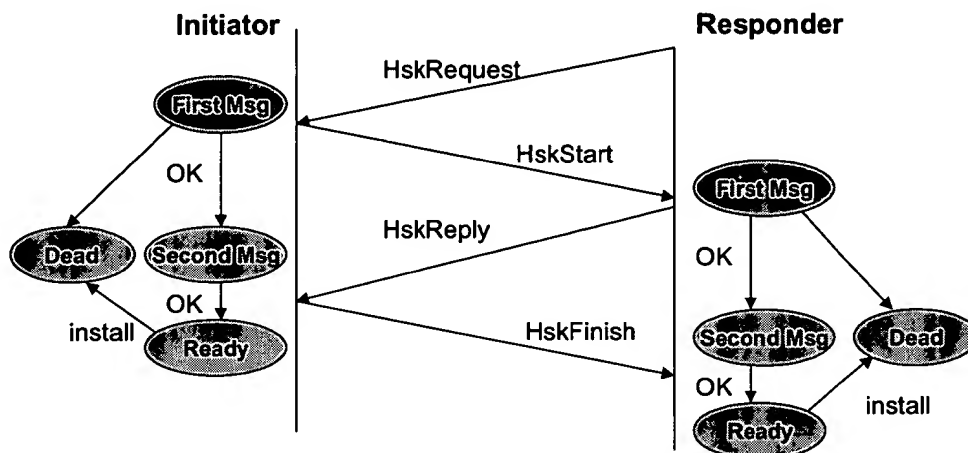


Figure 10 shows the key agreement interactions and corresponding state machines of the handlers that control these interactions. These state machines are embedded in the state SET_UP in Figure 9. Sub-states READY and DEAD are not necessarily related to the ones in Figure 9. For example, killing the handler does not mean that the session will die. It is perfectly normal that the handler will terminate when the key agreement finishes successfully, and the cipher suite gets “instantiated” in the session.

A quick summary of the messages sent during the handshake:

- **HskRequest**: a request from either party to re-negotiate a session
- **HskStart**: a request (or acknowledgement) from the client to the server to start a session. It contains the elliptic curve and cipher suite list suggested, the SPI at this end, a hint on the tags that the server should prove, a hint on the operations that we want to perform, and a public Diffie-Hellman (DH) key.
- **HskReply**: a reply to the previous HskStart from the server to the client. It contains the cipher suite chosen, the SPI at the server end, whether the server is a relay, certificates to prove the requested tags, a hint of tags that the client need to prove, a public DH key and a signature.

- We made the Initiator and Responder state machines similar by always introducing a HskStart message. If the protocol is started by the Initiator the HskStart is generated locally. Otherwise it will be generated by the Responder and transmitted through the network. We pay attention to alerts and “incorrect” messages that have valid random SPI, so we can end the session at any time during HandShake.

This term is used for the set of instances of encryption/decryption, hashing and authentication algorithms, along with the particular key-values, to be employed in a session. The protection layer is established through the HandShake phase.

$$g[U, f(V)] = g[V, f(U)] = z$$

142

Key generation algorithm

For details of key-generation see [*Ferguson*, Section 4.]. Four fixed byte-arrays are used as follows:

```
array A - for the key  $K_A$  that authenticates client to server
array B - for the key  $K_B$  that authenticates server to client
array C - for the key  $K_C$  that encrypts client's messages to
server
array D - for the key  $K_D$  that encrypts server's messages to
client.
```

The keys are symmetric, so K_C is used to decrypt messages from the client, as well as to encrypt them; likewise K_D for messages from the server. K_A is used both to create and to verify the client's HMAC's, and K_B for the server's.

The keys are derived from the byte-arrays as follows. The shared secret Z (or a function of it) is expressed in a fixed byte-array. This is appended to each of arrays A to D, giving byte-arrays which we will call Key_array A, B etc. For example, Key_array A = array A | Z .

The hash function h is applied recursively to each Key_array. From the Key_array s , byte-arrays s_0, s_1, \dots, s_k are created, obeying:

$$\begin{aligned} s_0 &= h(s) \\ s_n &= h(s_0 \mid s_1 \mid \dots \mid s_{n-1} \mid s) \quad (n \geq 1) \end{aligned}$$

Repeated recursion generates longer and longer byte-arrays as the argument to h . There are two target lengths: L_A for the authentication keys K_A and K_B , and L_E for the encryption keys K_C and K_D , determined by the corresponding algorithms. For each key, recursion proceeds till the length of $s_0 \mid s_1 \mid \dots \mid s_k$ equals or exceeds the target length. The first L_A or L_E elements make up the key.

Cipher suite support

The encryption algorithms currently supported are Blowfish with a 128 bit key, and triple DES with three independent keys for encryption. Blowfish is the recommended cipher because of its speed but 3DES is the conservative choice. Also only CBC mode and PKCS 5 padding is supported.

Our current hash algorithm is SHA-1 and our MAC algorithm is an HMAC construction based on SHA-1.

SPKI certificates are signed/verified using ElGamal with a 768-bit key as default. RSA with a 1024 bit key can be used if the client prefers. (All the named algorithms are explained in [*Schneier*].)

Authentication of messages

See [*Menezes, van Oorschot and Vanstone* p.355] and [*Ferguson*, Section 4.]

A hash is generated from the appropriate authentication key (K_A or K_B above), selected fields of the PDU header, and the PDU body. This is the MAC: it has a fixed length (of 20 bytes in our implementation). It is pre-fixed to the PDU body. If the encoding type is SECURE_DATA, the resulting string (MAC + PDU body) is encrypted using K_C or K_D . The resulting encrypted string we call Q. The PDU transmitted will consist of:

- PDU header
- PDU body + MAC if type is PROTECTED_DATA
- Encrypted string Q if type is SECURE_DATA

For the recipient to check the message, if the type is SECURE_DATA, it begins by decrypting Q, using K_C or K_D as before. This gives a string: MAC + PDU body. The selected header fields, the PDU body (without the MAC) and the authentication key K_A or K_B are input to the same hash algorithm to generate a second MAC. This is compared with the MAC received in the message: if they are unequal, the message is not authentic.

The PDU header fields omitted from the MAC derivation are:

toAddress, fromAddress, route.

Handling alert messages

During the key agreement protocol the default is to pay attention to non-authenticated alert messages that have the correct random SPI. These identifiers are sent in clear, so if the attacker listens to all our traffic and sends fatal alerts with the right SPI before the other party responds, it will still stop the session set-up. However, this attack would be much easier if we had a predictable SPI. The attacker could just flood the system with fatal alerts with typical SPIs. In SLS this problem is more evident than in SSL, because of the independence of the session from the transport. In this case we cannot make the assumption that it takes some effort to hi-jack the transport, as is the case for a TCP socket in SSL.

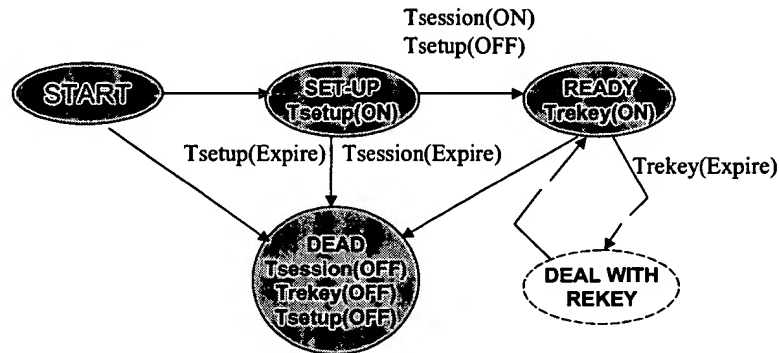
If we want to disable this type of attack completely, we could ignore all non-authenticated alerts and rely on timeouts to close failed sessions. This is not the default. The convenience of quick and detailed notification of session set-up failure is believed to be more important than countering an impractical denial-of-service attack. The attack can always be done at transport level anyway.

In any case, after a session is established only authenticated alerts are respected. At that point many messages with the SPIs in clear have been exchanged, and the randomization does not help much.

Alert messages can be fatal, forcing the other end to close the session, or warning, that in our first implementation are just logged. We support both internal alerts, generated as a side-effect of a PDU handling, and external alerts, those explicitly sent by the client. It is recommended however to avoid sending external alerts and rely on internal ones as much as possible. The alert codes used in SLS are described above (see "Alert" on page 122).

Timeouts description

Figure 11 Session timeouts and state transitions



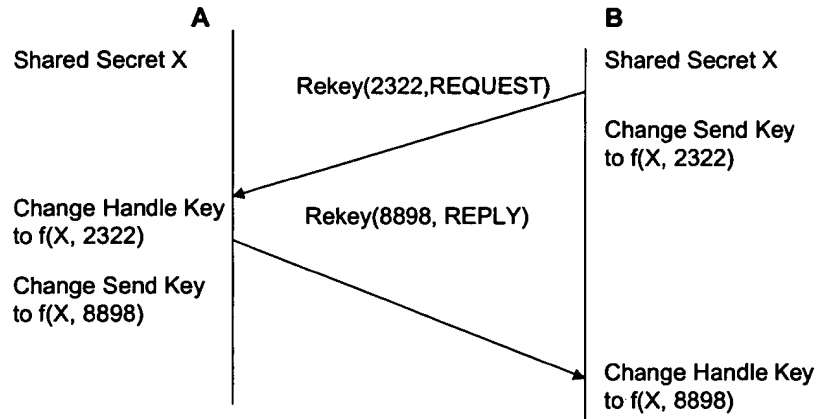
There are three built-in timeouts associated with a session:

- **Tsetup**: sets the maximum time taken by the key agreement protocol used in the SET_UP state. After that time the session becomes DEAD.
- **Tsession**: sets the maximum life expectancy of session. This value is the minimum of a fixed value (common for all sessions), and the life-expectancy of the certificates negotiated during the key agreement. After that time the session becomes DEAD.
- **Trekey**: sets the maximum time allowed before forcing a rekey operation on the session. Rekeying is currently not supported, and Trekey is set to infinity.

Figure 11 describes the behavior of the timeouts in relation to the session states. Tsetup sets a limit on the time spent on the SET_UP state, but it is reset after a transition to the READY state. Tsession limits the maximum time spent on the READY state. When Trekey expires we initiate the rekey and reset the timer, but this does not imply a state transition. Clearly, Trekey is only useful if it is smaller than Tsession, otherwise the session will never re-key.

Re-keying (not currently supported)

Figure 12 Rekey protocol.



After the session has been established it is possible to change the key used in the cipher and MAC operations by sending a REKEY message. However, this new key has to be based on the original shared secret negotiated using Diffie-Hellman (we do not re-run the key agreement protocol). Therefore, the re-key operation does not extend the life of the shared secret, only of the derived keys. In particular, the new key is obtained by exclusive-or of the first four bytes of the shared secret with a random integer before re-running the key generation algorithm. The one-way function used in that algorithm ensures that it is difficult to guess the next key, even if you know the previous one. The integer xor-ed with the shared secret is transmitted inside the REKEY message.

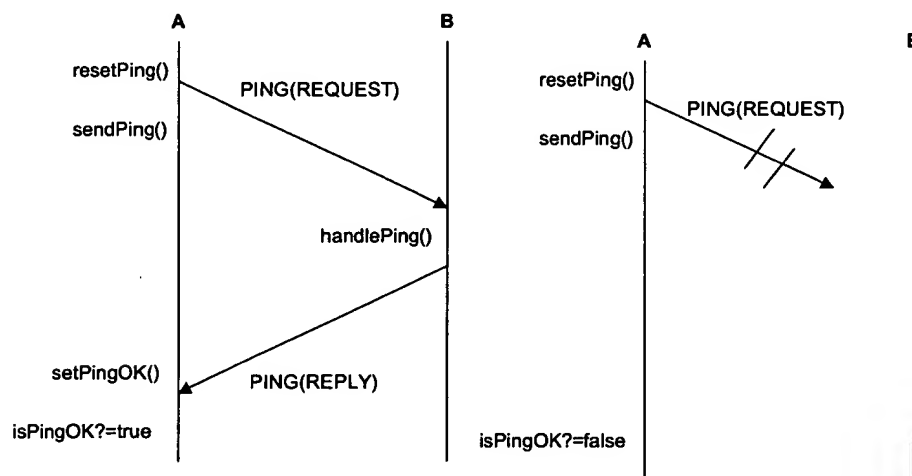
Figure 12 shows the basic protocol to re-key a session. Node B decides that it wants to start a rekey, so it generates a random number (2322) and sends a REKEY message with it. After that it re-keys the "send part" of its cipher suite right away, so the next message sent will be encrypted with the new key. When the REKEY message arrives to node A this node changes its "handle part" of the cipher suite to the new derived key. Then it checks that the message is a request and not a reply (the rekey was not initiated by him) and sends a REKEY reply message with possibly a different random integer (8898), changing the "send part" of its cipher suite too. When the reply message arrives to B, this node will update the "handle

part" of their cipher suite but it will not rekey the "send part" again because the message was tagged as "reply".

The important point of the protocol is that all the state is encoded in the messages. Provided that messages are not re-ordered, the receiver always has the right key to decrypt the message. It does not have to remember old keys or interrupt the service during re-keying. This avoids the need of an extra state in the protocol for re-keying.

Support for session scavenging

Figure 13 Ping protocol



SLS needs an external mechanism to detect that the other party in the session is no longer active. This is required because of the independence of transport and session "lives". We cannot assume that the underlying transport will detect that the other party abandons the session. The transport won't necessarily send a TCP keep-alive message, for example. We have to provide that service at a higher level.

Figure 13 shows the basic support provided for session scavenging. An external client can check whether the session is active by forcing its endpoint to send a PING message and resetting a flag that indicates a reply ping arrived. If the reply ping arrives the flag is set. After a certain time the client checks whether the flag is set, indicating whether the other end is still alive or not.

Support for tunneling

In SLS tunneling a PDU contains in its payload another PDU, so messages are sent using another SLS session as "transport". When the initiator sends the first protocol message, (HandShakeStart in the current implementation), the responder might reply that it is not the final end-point, so it cannot prove what was requested, and it wants to be a relay instead. At that point the client can decide whether to continue the key agreement or not. If it continues it will get a ready session that was negotiated as a "relay" and it can use that session to negotiate another one to the end-point. If this new end-point also wants to be a relay the process repeats. Typically the maximum depth of session nesting is limited to a fixed value to avoid a denial-of-service attack. As implemented, the Session objects are stacked. An initial session (A) is connected to the transport; a session B nested in A is "connected" to A, and so on. However arranged, it should be transparent to the client.

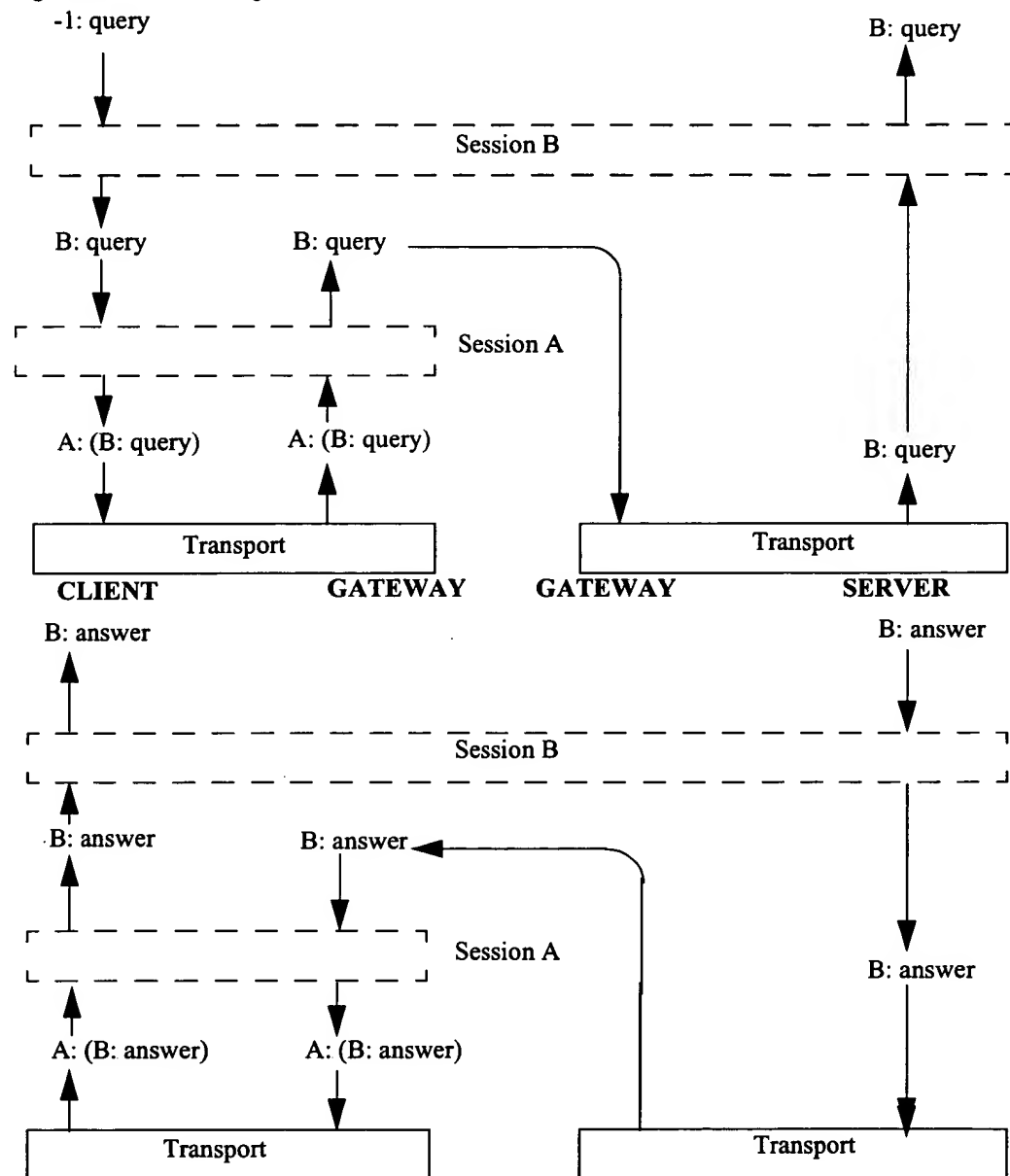
Figure 14 shows how to send messages from client to server and back after a nested session via an SLS gateway has been established. The diagram shows sender SPIs as initial characters with a colon, such as "A:" or "B:". The wrapped PDUs are bracketed. When a PDU passes through a session a layer of encryption is applied or removed. This is not shown in the diagram.

A session will perform automatic wrapping of a PDU inside another PDU while sending when:

- the sender SPI is valid (>0)
- the sender SPI does not match the one the session is going through (we use sender and not receiver SPI because the receiver one is not guaranteed to be unique).

In the current implementation, a PDU to be tunneled is initialized with an invalid sender SPI of -1. So session B just changes the sender SPI to "B". When the PDU is passed to session A, the conditions for wrapping are satisfied. During the wrapping the addresses of the inner PDU are copied into the header of the external PDU. This is given the message-type TUNNEL. The resulting TUNNEL PDU gets unwrapped when it is handled by session A in the gateway. The reply is wrapped by session A in the gateway, and unwrapped by session A at the client end. This is default behavior that can be overridden by a custom handler.

Figure 14 Tunnelling SLS sessions



- It can't be assumed that a message received in a normal session was not tunneled.
- Tunneled messages, which may have several layers of wrapping and encryption, can not be assumed to have traversed a firewall, and are not necessarily any more secure than normal messages.

Two e-speak cores can exchange core-managed resources and resource metadata. The exchanges comprise import and export of resources. They are needed for several reasons, including the following.

- A resource cannot be discovered in a vocabulary on an e-speak core unless the vocabulary has been registered on that core. If the vocabulary was created on another e-speak core, it is registered by importing it.
- A resource cannot be registered in a contract on an e-speak core, unless the contract itself has been registered on that core. If the contract was created on another e-speak core, it is registered by importing it.
- Resource metadata can be imported into an e-speak core, to cache it. This will make lookup of those resources faster.

Two core-managed resources handle communication between e-speak cores.

- 151

- The Remote Resource Manager (RRM) is responsible for managing metadata: importing and exporting resources from the remote e-speak core. The ESName for the RRM on any given e-speak core is:
es://<server>/CORE/RemoteResourceManager

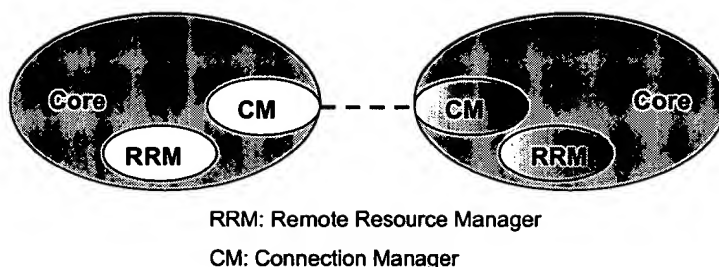


Figure 15 Core-core communication components

Connection Manager

The Connection Manager provides the core-to-core connection handling APIs. Each connection is associated with a Protection Domain, Outbox and an Inbox. The inbox and outbox along with the Router form the message forwarding subsystem to the remote core (see Figure 7).

The core-to-core connection can be a secured channel: SLS messages may be exchanged to set up a secure channel.

```
class connectionManager{
public synchronized String openConnection(String coreUrl) 1
throws UnknownHostException
public synchronized void closeConnection(String conID)
throws UnknownHostException
public synchronized CMArg closeConnectionFromRemote(CMArg
cmArg)
throws UnknownHostException
public synchronized ESArray getConnections()
}
```

¹ The coreUrl should be type ESName, in the current implementation it is type String.

CM Methods: openConnection()

The openConnection() invocation is synchronous. The caller has to wait until the openConnection() returns or times out (the current default time out period is 10 seconds). The parameter is the URL of the remote core's root frame: a URL of the form es://host

When the Connection Manager executes this function it sends a PDU message to the remote core. The toAddress of the PDU is set to es://host/CORE. (Note that only "es://host" is the argument passed by the caller of openConnection().) The PDU is a messageForResource with a payload type of METHOD_CALL: the payload field is an instance of methodCall. The methodCall.interface is "Core" and the method field is "bootstrap". The fromAddress is set to es://<localCore>/CORE, where <localCore> is the host and port for the Connection Manager's e-speak core.²

The messageType field of PDU is set to HANDSHAKE for this request in the current implementation. It should, for consistency, be APPLICATION_MESSAGE.

Once the message has been sent, the Connection Manager waits for a reply PDU message. The remote core receiving the message replies with a bootstrapReply message (see "Initial Connection to the Core" on page 130). In the current implementation this reply is ignored.

Next the Connection Manager sends a "negotiate" message to the remote core. This consists of an empty messageForResource instance (payloadType is set to OBJECT and the payload contains the null object). The toAddress of the PDU is set to es://host/CORE/ConnectionManager. The fromAddress of the PDU is set to es://<localCore>/CORE/ConnectionManager.

Once this message is sent the Connection Manager waits for a reply from the remote Connection Manager. In the current implementation this reply is ignored.

The returned value of openConnection is a String denoting the server and port of the remote core to which a connection has been made. Thus a connection to es://foo.bar.com:8000/ returns a String: "foo.bar.com:8000". This String can be used to identify the connection for later connection management operations.

² In the current implementation all URLs created by the connectionManager begin with "tcp://" instead of "es://"

CM Methods: proposed future negotiation

1. Initiator builds a negotiation proposal, and sends the proposal to the remote core. The offer includes parameters such as Core Version, e-speak version and PDU size (for buffering, fragmentation and reassembly).

3.The initiator then builds the agreed upon offer and sends the final offer to the remote core.

CM Methods: closeConnection()

The parameter `conID` to `closeConnection()` is the String previously returned from `openConnection()`. When `closeConnection()` is invoked the Connection Manager sends a PDU containing a `messageForResource` instance to the remote Connection Manager. The `MessageForResource` has a `payloadType` of `METHOD_CALL`. The `interfaceName` is "ConnectionManagerInterface" and the `methodName` is "closeConnectionFromRemote". The parameter `CMArg` is defined as follows:

```
class CMArg { (ESF)
String localURL;
String remoteURL;
int type; // CLOSECONNECTIONREQUEST=1 CLOSECONNECTIONREPLY=2
}
```

The localURL field is set to the host+server port for the sending core for example: "initiator.bar.com:8080". The remoteURL field is set to host+server port for the remote core. The type is set to CLOSECONNECTIONREQUEST. When the remote core receives this message, it can use the localURL, remoteURL pair to identify the connection. The remote core sends a messageForResource with a payloadType of METHOD_RESULT. The Ob field of the MethodResult class is an instance of CMArg. The localURL and remoteURL are unchanged, but the type field is set to CLOSECONNECTIONREPLY. Having sent this, the remote core closes the connection. When the initiating core receives this reply, it closes the connection.

CM Methods: getConnections

The function getConnections() returns the state of current connections. Each element of the returned ESArray is a String of the form by the IP address of the remote host and the port number on which the remote e-speak core is located separated by a colon, e.g.: "host.foo.com:8000"

Remote Resource Manager

The Remote Resource Manager (RRM) handles metadata related functions. It provide the capability to export and import resources to and from remote e-speak cores.

RRM Message Class

Instances of the class payloadForRRM are used as arguments or returned by the RRM:

```
class payloadForRRM{ (ESF)
  int payloadType;
  boolean topLevel;
  int importExportMode;
  byte[] contextPDU;
  ESArray resourceTable;
  ESArray tablesArray;
}
```

The following are permissible values for payloadForRRM.payloadType:

```
EXPORT_REQUEST=0; //Export Request
EXPORT_REPLY=1;   //Export Reply
```

```

IMPORT_REQUEST=2; //Import Request
IMPORT_REPLY=3;   //Import Reply
UPDATE_EXPORTED_RESOURCE_REQUEST=4; //Update Export Resource
Request
UPDATE_EXPORTED_RESOURCE_REPLY=5; //Update Export Resource
Reply
UPDATE_IMPORTED_RESOURCE_REQUEST=6; //Update Import Resource
Request
UPDATE_IMPORTED_RESOURCE_REPLY=7; //Update Import Resource
Reply
UNEXPORT_REQUEST=8; //Unexport Resource Request
UNEXPORT_REPLY=9;   //Unexport Resource Reply
IMPORT_ERROR=10;    //Import Error
EXPORT_ERROR=11;    //Export Error

```

RRM Class:

```

class RemoteResourceManager{
void exportResource(ESName esname, boolean topLevel, int mode,
String server)
throws NameNotFoundException, StaleEntryAccessException

PayloadForRRM importResourceFromMsg(PayloadForRRM rrmPayload)
throws RemoteException

void importResource(ESName esname, boolean topLevel, int type,
String server)
throws ImportFailedException

PayloadForRRM exportResourceAsMsg(PayloadForRRM rrmPayload)
throws StaleEntryAccessException, NameNotFoundException,
RemoteException

void unExportResource(ESName esname, String server)
throws RequestNotDeliveredException

PayloadForRRM unExportResourceFromMsg(PayloadForRRM rrmPayload)
throws NameNotFoundException,
StaleEntryAccessException,
QuotaExhaustedException,
InvalidNameException,
PermissionDeniedException,
RemoteException

void updateExportedResource(ESName esname, boolean topLevel, int
mode, String server)

```

```
throws StaleEntryAccessException,
NameNotFoundException,
RequestNotDeliveredException,
ExportFailedException
```

```
PayloadForRRM updateExportedResourceFromMsg (PayloadForRRM
rrmPayload ,
String fromServer )
    throws NameNotFoundException,
           StaleEntryAccessException,
           PermissionDeniedException,
           InvalidValueException,
           UpdateFailedException,
           RemoteException
```

```
void updateImportedResource (ESName esname, boolean topLevel, int
type, String server)
throws NameNotFoundException,
StaleEntryAccessException,
RequestNotDeliveredException
```

```
PayloadForRRM updateImportedResourceFromMsg (PayloadForRRM
rrmPayload)
throws RemoteException
```

```
void exportOnConnecting (ESName esname, boolean toplevel, int
type)
throws NameNotFoundException;
}
```

RRM Methods: exportResource()

```
void exportResource (ESName esname, boolean topLevel, int mode,
String server)
throws NameNotFoundException, StaleEntryAccessException
```

The function exportResource exports the resource identified by esname to the server identified by server. The server parameter is a String of the form hostname:port, for example "foo.bar.com:8080". The boolean topLevel indicates whether this is to be a recursive export (topLevel = false) or not (topLevel = true). A recursive export will export all resources that are referenced in the metadata of the resource identified by esname (vocabularies, contracts and the like). The mode parameter indicates whether this is to be export by reference or export by value. Export by reference copies the metadata but not the resource state. Any invocation

RRM Methods: `importResourceFromMsg()`

The function `importResourceFromMsg` is invoked by a remote RRM (B) to tell RRM (A) to import the resource(s) contained in the argument `rrmPayload`, an instance of `payloadForRRM`. The `payloadType` field is set to `EXPORT_REQUEST`. The `topLevel` field is set to `false` if the import is to be recursive. The `ImportExportmode` field is set to `BY_VALUE` (0) or `BY_REFERENCE` (1). The `contextPDU` is not used by RRM A; it is returned to RRM B. The intent of this field is to enable RRM B to identify the context (typically an attempt to send a message) that caused it to invoke `importResourceFromMsg`.

Each element of `rrmPayload.tablesArray` is itself an `ESArray`. There is an `ESArray` in `tablesArray` corresponding to each element in `resourceTable`. Each `ESArray` in `tablesArray` consists of four elements.

- short typeCode
- ResourceSpecification spec

- ResourceDescription desc
- Object resource

The following are permissible values for typeCode:

```

INBOX_CODE = 0
META_RESOURCE_CODE = 1
PROTECTION_DOMAIN_CODE = 2
RESOURCE_FACTORY_CODE = 3
CONTRACT_CODE = 100
CORE_DISTRIBUTOR_CODE = 110
IMPORTER_EXPORTER_CODE = 120
MAPPING_OBJECT_CODE = 140
NAME_FRAME_CODE = 150
REPOSITORY_VIEW_CODE = 160
SECURE_BOOT_CODE = 170
SYSTEM_MONITOR_CODE = 180
VOCABULARY_CODE = 190
CORE_MANAGEMENT_SERVICE_CODE = 200
DEFAULT_VOCABULARY_CODE = 210
DEFAULT_CONTRACT_CODE = 220
FINDER_SERVICE_CODE = 230
CONNECTION_MANAGER_CODE = 240
REMOTE_RESOURCE_MANAGER_CODE = 250
EXTERNAL_CODE = 1000
EXTERNAL_RESOURCE_CONTRACT_CODE = 1001

```

ResourceSpecification and ResourceDescription are defined in Chapter 3, "Resource Data, Searches & Vocabularies".

The resource field is omitted if the resource identified by the ESName in resourceTable is an external resource (typeCode = EXTERNAL_CODE). Otherwise the resource field is an instance of the Core-managed resource specified. It will contain the data members defined for this Core-Managed Resource type in Chapter 4, "Core-Managed Resources". The resource field is included even if the export mode is BY_REFERENCE.

RRM Methods: importResource()

```

void importResource(ESName esname, boolean topLevel, int type,
String server)
throws ImportFailedException

```

This method instructs the RRM to import the named resource from the server identified by the String server (the format of this String is host:port). The boolean topLevel is set to false if the import is to be recursive. Permissible values of the argument type are BY_VALUE or BY_REFERENCE. When this function is invoked on the RRM it sends a message to the remote RRM on the core denoted by the server parameter. This is a messageForResource, containing an instance of methodCall with interfaceName: "RemoteResourceManagerInterface", methodName: "exportResourceFromMsg" and a single element in the argument array, of type payloadForRRM. The payloadType for payloadForRRM is IMPORT_REQUEST. The importExportMode is set to BY_VALUE or BY_REFERENCE. The contextPDU field is used by the RRM to identify the context for the reply to this message. Typically it contains a serialized PDU. The resourceTable contains a single element: the esname parameter passed in the call of importResource.

RRM Methods: exportResourceFromMsg()

```
PayloadForRRM exportResourceFromMsg(PayloadForRRM rrmPayload)
throws StaleEntryAccessException, NameNotFoundException,
RemoteException
```

The function exportResourceFromMsg is called from a remote RRM in response to an invocation of importResource on the remote RRM. The rrmPayload parameter contains the data defined in the description of importResource. The returned PayloadForRRM is sent in a messageForResource which contains an instance of methodResult. This PayloadForRRM has type IMPORT_REPLY. The importExportmode, topLevel and contextPDU fields will be those contained in the original rrmPayload. The resourceTable and tableArrays contains the list of ESNames of resources, and their metadata and state as described in the description of importResourceFromMsg.

RRM Methods: unExportResource()

```
void unExportResource(ESName esname, String server)
throws RequestNotDeliveredException
```

The function unExportResource causes the RRM to try to unexport the resource from the remote e-speak core identified by server (format "host:port"). It does this by sending an instance of methodCall with interfaceName "RemoteResourceManagerInterface", methodName "unExportResourceFromMsg" and a single element in the argument array of type payloadForRRM. The payloadType for payloadForRRM is UNEXPORT_REQUEST. The topLevel and

RRM Methods: unExportResourceFromMsg()

The `unExportResourceFromMsg` is invoked from a remote RRM in response to a call of `unExportResourceFromMsg` on the remote RRM. The `rrmPayload` contains the data defined in the description of `unExportResourceFromMsg`. The intent is that the RRM receiving an invocation of `unExportResourceFromMsg` should remove the resource contained in `rrmPayload` from its repository. The `PayloadForRRM` instance returned contains a `payloadType` of `UNEXPORT_REPLY` and the `contextPDU` passed in the `rrmPayload` parameter. All other fields are unused.

```
void updateExportedResource(ESName esname, boolean topLevel, int
mode, String server)
throws StaleEntryAccessException,
NameNotFoundException,
RequestNotDeliveredException,
ExportFailedException
```

161

RRM Methods: updateExportedResourceFromMsg()

```

PayloadForRRM updateExportedResourceFromMsg (PayloadForRRM
rrmPayload ,
String fromServer )
    throws NameNotFoundException,
           StaleEntryAccessException,
           PermissionDeniedException,
           InvalidValueException,
           UpdateFailedException,
           RemoteException

```

The function `updateExportedResourceFromMsg` is invoked by a remote RRM when it wishes to update resources which have been exported previously. The intent is that the RRM receiving the call of this function replaces the metadata (and the state in the case of an export `BY_VALUE`) of each resource in the `resourceTable` in `rrmPayload` with the metadata and state contained in `tablesArray`. Only resources that have already been registered will have their metadata and state updated. The `PayloadForRRM` returned as the result will have the `contextPDU` of the `rrmPayload` parameter and a `payloadType` of `UPDATE_EXPORTED_RESOURCE_REPLY`. All other fields will be ignored by the remote RRM that invoked `updateExportedResourceFromMsg`, when it receives this result.

RRM Methods: updateImportedResource()

```
void updateImportedResource(ESName esname, boolean topLevel, int
type, String server)
throws NameNotFoundException,
StaleEntryAccessException,
RequestNotDeliveredException
```

The `updateImportedResource` function is similar to the `importResource` function. The major difference is that the RRM has previously imported the resource identified by `esname`. The RRM will invoke the `updateImportedResourceFromMsg` on the remote RRM identified by the String `server (host:port)`. The `PayloadForRRM` passed as a parameter in `updateImportedResourceFromMsg` has `payloadType` of `UPDATE_IMPORTED_RESOURCE_REQUEST` and will have a single element in `resourceTable`: the `ESName` of the resource that needs updating. Note that this may not be the same `ESName` that is received as a parameter to

RRM Methods: updateImportedResourceFromMsg()

The function `updateImportedResourceFromMsg` is invoked by a remote RRM when it has received an invocation of `updateImportedResource` and needs to update a resource's metadata (and possibly state). The `rrmPayload` will contain the data described in the description of `updateImportedResource` above. The `PayloadForRRM` returned from the `updateImportedResourceFromMsg` function has a `payloadType` of `UPDATE_IMPORTED_RESOURCE_REPLY`. The `topLevel`, `importExportMode` and `contextPDU` fields will be the same as in the `rrmPayload` parameter. The `resourceTable` field and `tablesArray` respectively contain the `ESnames` and metadata (and possibly state) of the resources to be updates. Note that even though a single resource `ESname` is all that is contained in the `rrmPayload` parameter, the result can contain many `ESnames` if the `topLevel` flag is set to `false` (indicating recursive import/export).

The resource is added to the list of resources to be exported when connection is established. The parameter `esname` is the `ESName` of the resource to be exported.

The parameter type indicates the type of export. It can take the value BYVALUE (0) or BYREFERENCE (1).

The following Core-managed Resources cannot be exported or imported.

- Developer Release X.03.03.00, September 2000**

- Table 10 Core-managed Resource export restrictions**

Resource	Pass by reference restrictions
name-frame	Cannot be used as a component of an ESName sent to the Core for name resolution
Repository View	Cannot be used in a Search Recipe
Resource Contract	Cannot register a Resource in this Contract
Vocabulary	Cannot be used in a Search Recipe

The initiating core also performs similar clean up process. The Protection Domain, Outbox assigned to the connection are removed.

ESNames

ESNames denote an access path to a resource. ESNames conform to the format and grammar defined for Universal Resource Identifiers (URIs) [see *RFC 2396*].

ESNames are therefore URIs and more specifically, since they denote the access path for the resource, ESNames are also Universal Resource Locators (URLs). ESNames have the following format.

```
es://<host>/<relative path>
```

The host part of an ESName is either the host name or the IP address of the host on which the e-speak core is located together with an optional port number. If the port number is not specified, the current implementation will throw an exception.

Discussions are underway with IANA for a standard port number to be assigned.

The full form of an ESName (es://<host>/<relative path>) is known as an absolute ESName. Subsets of this syntax also denote ESNames (see "ESName BNF" on page 169). However, it may not be possible to resolve such ESNames if we do not have the necessary context.

The relative path component of an ESName must be unique on the given e-speak core. The path is relative in the sense that it is given a global context by the <host> element of the ESName. If the <host> element is missing (e.g. es://path or es:/path), then the resolver must decide the global context in which to begin resolution. Usually the global context is assumed to be the current host.

The path consists of a set of Strings separated by "/", for example "a/b/c". The path is resolved by taking each String element in order and resolving that in the current name-frame. If this returns a name-frame the next element is resolved in that name-frame. The process continues until there are no more elements in the path in which case we have resolved the ESName to the intended resource. The first element will be resolved in the root name-frame of the e-speak core denoted by the server part of the ESName. For example, taking "a/b/c", "a" is resolved in the e-speak core's root name-frame to return a name-frame which we denote NF(a). Next b is resolved in NF(a), to return a name-frame which we denote NF(a b). Finally c is resolved in the name-frame NF(a b).

If a String element of the path component other than the final component fails to resolve to a name-frame, name resolution has failed. If the final element fails to resolve to a resource, name resolution fails.


```
/Core
/ContractContract
/VocabularyContract
```

Canonical ESName

```
es://<hostport>/proc/resource/<type_string>/<unique_id>
```

The <hostport> field is of the form host name or IP address followed by a port number separated by a ":" as specified in [RFC 2396].

The following are the permissible values of <type_string>, they denote the Resource Type (see Chapter 3, “Resource Data, Searches & Vocabularies”)

```
"Inbox"
"MetaResource"
"ProtectionDomain"
("ResourceFactory")
("ConnectionManager")
("RemoteResourceManager")
"Contract"
("CoreDistributor")
"ExternalResource"
"ExternalResourceContract"
("ImporterExporter")
("MappingObject")
"NameFrame"
"RepositoryView"
("SecureBoot")
"SystemMonitor"
("AccountManager")
```


ESName BNF

Here is the BNF for ESNames. Please refer to [RFC 2396] for any element not defined directly.

```

ESName = [ absoluteESName | relativeESName ] [ "#" fragment ]
absoluteESName = es ":" hier_part
relativeESName = ( net_path | abs_path | rel_path ) [ "?" query ]

hier_part = ( net_path | abs_path ) [ "?" query ]

net_path = "//" hostport [ abs_path ]
abs_path = "/" path_segments
rel_path = rel_segment [ abs_path ]

rel_segment = 1*( unreserved | escaped | ";" | "@" | "&" | "="
| "+" | "$" | "," )

hostport = host [ ":" port ]
host = hostname | IPv4address
hostname = *( domainlabel "." ) toplabel [ "." ]
domainlabel = alphanum | alphanum *( alphanum | "-" ) alphanum
toplabel = alpha | alpha *( alphanum | "-" ) alphanum
IPv4address = 1*digit "." 1*digit "." 1*digit "." 1*digit
port = *digit

path_segments = segment *( "/" segment )
segment = *pchar *( ";" param )
param = *pchar
pchar = unreserved | escaped | ":" | "@" | "&" | "=" | "+" | "$"
| ","
query = *uric
fragment = *uric

```


PDU Marshalling format

A PDU is transmitted as a 32 bit length (in network byte order), followed by the buffer containing the PDU itself.

All data inside a PDU is marshalled in network byte order.

- int is a 32 bit integer
- long is a 64 bit integer.
- short is a 16 bit integer
- char is a 16 bit character
- boolean is marshalled as a single byte (0x01) for true (0x00) for false.
- String is marshalled as a 16 bit length followed by each character in the String as 16 bits per character
- byte[] is marshalled as a 32-bit length followed by the bytes.
- ADR are marshalled by converting them to ASCII canonical s-expressions defined in "SPKI BNF Formats" (Chapter 5, "Access Control") and then marshalled as a byte array using marshalBytes.

E-speak Serialization Format

The basic types recognized are byte, short, int, long, float, double, and string. The four integral types byte, short, int, and long are 1, 2, 4, and 8 bytes long respectively, and are always sent most significant byte first. The float and double types are sent just as in Java. The string type is intended to contain text rather than arbitrary binary data, and the text must be a valid UTF-8 encoded string as per [RFC 2279]. It is hoped that "string" will not be confused with java.lang.String.

We also recognize arrays of types. The type foo[] is sent as a length followed by that many instances of type foo. If the length is -1, then a NULL is returned. If the length is 0, an empty array is returned. Otherwise an array with that many elements is returned.

A map is sent in the same syntactic way as an array, but there is an implicit Key/value association between pairs of elements; all the evenly indexed elements (0, 2, etc.) are Keys, and all the odd indexed elements are values. Some maps may allow multiple occurrences of the same Key.

The length field is encoded in a single byte if the value of the length is -1..62 inclusive; the encoding is 129 more than the length. Thus, -1 is sent as the byte value 128, and a length of 3 is sent as the byte value 132. Lengths from 63.. $2^{31}-1$ are sent as a 4 byte integer. Lengths below -1 or greater than $2^{31}-1$ are illegal at the present time.

All elements are sent as a signal byte that indicates the type of the object that follows, followed by the data for that object.

Signal bytes are entirely single bytes. They are encoded by literal ASCII characters (e.g., A), literal ASCII characters but with the high byte set (char)('J'+128)).

Here is the Backus-Naur Form (BNF) for "Ob" an object serialized in the e-speak format, using the signal bytes defined currently.

```
Ob = 'E', <RSD >|
    'D', <ResourceDescription >|
    'S', <ResourceSpecification >|
    '[', <ESUID >|
    'c', <SearchPredicate >|
    'C', <SearchRecipe >|
    ']', <VocabularyDeclaration >|
    ',', <Preference >|
    '~', <FilterSpec >|
    'T', <AttributeProperty >|
    'A', <AttributePropertySet >|
    'a', <Attribute >|
    'B', <AttributeSet >|
    'V', <Value >|
    'Y', <ValueType >|
    'F', <ESName >|
    ')', <ESString >|
    'q', <AttributePredicate >|
    '<', <NamedObject >|
    '=', <ProfileAttributeSet >|
    '>', <UserProfile >|
    'N', <NameSearchPolicy >|
    '.', <FinderResults >|
    '/', <FinderContext >|
```

09-7687

The components for each of the remaining non primitive type are defined in the relevant sections of this specification (**to do: need to pull these definitions in to complete the BNF**).

In the following BNF, the meta-symbol => means "is sent as." The convention is as follows:

```
String    => string
Integer   => int
Long      => long
Boolean   => byte
Null      =>
ByteArray => byte[]
ObjectArray => Ob[]
ESMap     => map
ESArray   => Ob[]
ESSet     => Ob[]
ESList    => Ob[]
```

ESMap, ESArray, ESSet and ESList in the current implementation are marshalled using Java serialization.

00000000000000000000000000000000

- NOTE:** For all RFC's, access www.ietf.org

- 174

Chapter 7 Exceptions

E-speak defines a set of *exceptions* to inform Clients when an error occurs in the system. Two classes of exceptions are defined: *run-time exceptions* and *recoverable exceptions*.

Run-Time Exceptions

Run-time exceptions are thrown when programming errors occur. A program catching such exceptions may terminate. `ESRuntimeException` has the following subclassed exceptions:

- `CorePanicException` is thrown when the Core is unable to process the request. Although the Core attempts to notify all Clients of its inability to continue operating, it also replies with this exception for as long as it can. The Core can continue to accept new messages as the problem may be limited to the execution of a single message.
- `ServicePanicException` is thrown when a service is unable to process the request. This can be a terminal error for the service, in which case the service exits. Or it can simply mean that the request being processed caused an internal error that was not recoverable, and the service accepts new requests.
- `RepositoryFullException` is thrown when the request attempted to add additional information to the Core's Repository, but the Repository was full. This exception can be recovered from if the Client is able to delete one or more Resources from the Repository. It is a run-time exception because almost every message can possibly throw this exception, and the Client has no guaranteed recourse (because some other application can consume the Repository space freed up by this Client).

- `OutOfOrderRequestException` is thrown when the state of the system is inconsistent with the request.
- `ConnectionFailedException` is thrown when the Connection Manager fails to establish a connection, details are contained in the exception state.
- `InvalidParameterException` is thrown by any other programming errors. This exception has three subclasses:
 - `NullParameterException` is thrown where a null parameter was supplied but is not allowed. This error is often caused by passing an uninitialized object.
 - `InvalidValueException` is thrown when a parameter is outside the allowed range.
 - `InvalidTypeException` tells the programmer that the name specified is bound to the wrong type of Resource.

Recoverable Exceptions

Recoverable exceptions occur due to a problem with the state of the system. For example, when the Client sends a message to request access to a Resource, the message may be undeliverable, perhaps because the Handler's Inbox is full. Recovery for this case can be as simple as resending the message.

The base exception is `ESException`. This exception is subclassed into three major categories: `ESLibException`, `ESInvocationException` and `ESServiceException`.

`ESLibException` is the base class for client library exceptions. It should not be thrown itself but rather a subclass exception. Currently one subclass is defined.

- `CoreNotFoundException` indicates that a core could not be found to connect to. Either change the specification of the core or insure the core is running to correct this exception.

`ESInvocationException` is a base class for all the exceptions that can be thrown by the Core back to the Client occurring during the processing of the request. Exceptions thrown by most handlers are included here to reduce the number of explicit classes of exceptions that must be caught. This exception is further subclassed into:

`NamingException` results from a wide variety of problems. Regardless of the cause, this exception, or any of its subclasses, is thrown only for the primary Resource of the message header. Five subclasses are defined:

- `NameNotFoundException` is thrown when the name resolution process failed to find a given name. The Client can recover by changing `ESName`.
- `EmptyMappingException` is thrown when a Mapping Object is associated with the name, but that Mapping Object has no usable accessors. This condition arises when the accessor has no elements, the elements refer to unregistered Resources, or the Resources did not pass the visibility tests. The Client can recover by changing `ESName` or trying again with a different set of Keys.
- `UnresolvedBindingException` is thrown when all the accessors of the Mapping Object are search requests. The Client can recover by requesting a lookup using the search request.
- `MultipleResolvedBindingException` is thrown when the Mapping Object has more than one explicit binding.
- `LoopDetectedException` is currently unused.

`StaleEntryException` is thrown if the Resource no longer exists. The Core removes any stale handles from the Mapping Object before returning the exception. A retry does not result in this exception unless another referenced Resource has been unregistered.

`PermissionDeniedException` is thrown by any Resource Handler when the client is not authorized to access the Resource. The Client can recover by retrying with a different set of certificates. One subclass is defined

- `SessionRequiredException` is thrown when a client attempts to send it a message without first setting up a session. The service has security enabled and is performing access control checks. A secure session is needed so that the access control check can be made. This would normally be handled by the client

library and is transparent to the application programmer. The client recovers from this exception by exchanging SLS messages with the service to establish a session.

`QuotaExhaustedException` is thrown when the Client attempts to define more Resources than it is allowed as defined by the quota assigned. The Client can delete other Resources (thus freeing up quota) and reattempt the request.

`MethodNotImplementedException` is thrown when the Client attempts to invoke a method on a Resource that is not implemented even though the method is consistent with the type of the Resource. This is typically used to "stub-out" routines when a service is under development.

`RecoverableCoreException` is thrown when there is a problem while processing the request. There are two associated subclass exceptions:

- `RequestNotDeliveredException` is thrown when the Core never started processing the message. This exception can be thrown by the Client library if it implements time-outs or in by the Core if the corresponding queue is full. It may be possible to recover from this exception by resending the message.
- `PartialStateUpdateException` is thrown when the Core cannot finish processing the message. The Client may need to find out what state was changed before attempting recovery, for example, by examining the state of the metadata.

`TimedOutException` is thrown when a message being written to or received from a channel has not successfully completed within the caller defined time period.

`UndeliverableRequestException` is thrown when the message cannot be delivered to the Resource Handler. In the current implementation, this is not thrown with security enabled, the security subsystem silently ignores such messages (in case they are a denial of service attack) and the Client has to wait for a `TimeoutException`. There are two subclass exceptions of `UndeliverableRequestException`:

- `RecoverableDeliveryException` is due to temporary conditions such as a full Mailbox. Recovery can be as simple as retrying.

- ESServiceException** is a base class exception for all service-defined exceptions.

- ## Exception State

```
class ESEException {
    int errno;
    Object[] info;
}
```

```
NONE= 0
ESRuntimeExceptions (1-99 reserved)
INVALID_PARAMETER= 1
NULL_PARAMETER= 2
INVALID_VALUE= 3
INVALID_TYPE= 4
OUT OF ORDER REQUEST= 5
```

CORE_PANIC= 6
SERVICE_PANIC= 7
REPOSITORY_FULL= 8
 ESEExceptions (100-999 reserved)
INVOCATION= 100
NAMING= 101
NAME_NOT_FOUND= 102
EMPTY_MAPPING= 103
UNRESOLVED_BINDING= 104
MULTIPLE_RESOLVED_BINDING= 105
PERMISSION_DENIED= 106
QUOTA_EXHAUSTED= 107
STALE_ENTRY_ACCESS= 108
RECOVERABLE_CORE= 109
PARTIAL_STATE_UPDATE= 110
REQUEST_NOT_DELIVERED= 111
UNDELIVERABLE_REQUEST= 112
UNRECOVERABLE_DELIVERY= 113
RECOVERABLE_DELIVERY= 114
TIMED_OUT= 115
METHOD_NOT_IMPLEMENTED=116
LOOP_DETECTED= 117
SESSION_REQUIRED= 118
CONNECTIONFAILED= 119

E-speak defined service exceptions

SERVICE= 200

Name frame service exceptions

NAMEFRAME= 201
INVALID_NAME= 202
NAME_COLLISION= 203
LOOKUP_FAILED= 204

Import/Export service exceptions

REMOTE= 210

Client Library defined exceptions

ESLIB = 950
CORE_NOT_FOUND= 951

Exception numbers 1000+ are reserved for application use

008021" 2000/09/01

Exception hierarchy

Here is the exception hierarchy. Indentation indicates position in the hierarchy.

```

ESRuntimeException
  ServicePanicException
  OutofOrderRequestException
  CorePanicException
  ConnectionFailedException
  RepositoryFullException
  ConnectionFailedException
  InvalidParameterException
    InvalidTypeException
    InvalidValueException
    NullParameterException
ESEException
  ESLibException
    CoreNotFoundException
  ESServiceException
    ESRemoteException
    ESNameFrameException
    NameCollisionException
    InvalidNameException
    LookupFailedException
  ESInvocationException
    StaleEntryAccessException
    PermissionDeniedException
    SessionRequiredException
    QuotaExhaustedException
    MethodNotImplementedException
    RecoverableCoreException
    PartialStateUpdateException
    RequestNotDeliveredException
    NamingException
    MultipleResolvedBindingException
    UnresolvedBindingException
    NameNotFoundException
    LoopDetectedException
    EmptyMappingException
    TimedOutException
    UndeliverableRequestException
    RecoverableDeliveryException
    UnrecoverableDeliveryException

```


09303037 1E0300

Events

An Event is defined as follows:

Developer Release X.03.03.00, September 2000

EventAttributeSet contains an **AttributeSet** (xref to chapter on resourceDescriptions, section on Resource Description defines AttributeSet):

The string format indicates the format of each `Attribute` in the `AttributeSet` `attrs`. “VOCAB” means that the attributes have to be valid in the vocabulary references in the `AttributeSet` `attrs`. “SIMPLE” means the attributes are simple (name, value) pairs and no valid vocabulary is specified in `attrs`.

The Core is a Publisher of Events. All Events published by the Core go to a single service called the *Core Distributor Service*. This service is the Resource Handler for several Distributor Resources, each dealing with a Core-generated Event of a different type. These are:

- These types are used to maintain the coherence of metadata and the Resource state shared by value. Both are described in the *Base Event Vocabulary*.

Developer Release X.03.03.00, September 2000

The eventAttrs field consists of a set of name, string-value pairs. Two common examples are:

- name "Name", value is the stringified version of the ESUID of the Resource responsible for generating the event
- name "FailureDetail", value is a string indicating the nature of the failure

The format string of the EventAttributeSet eventAttrs is "SIMPLE".

The payload field is null for events generated by the e-speak Core.

The following is the list of prefix strings used by the current implementation.

```
"core.mutate.NameFrameInterface."
"core.mutate.MailBoxInterface."
"core.mutate.ProtectionDomainInterface."
"core.mutate.RepositoryViewInterface."
"core.mutate.VocabularyInterface."
"core.mutate.VocabularyToolBoxInterface."
"core.mutate.ResourceFactoryInterface."
"core.mutate.ResourceManipulationInterface."
"core.mutate.ImporterExporterInterface."
"core.mutate.SecureBoot."
"core.failure."
"core.failure.exception."
"notifySync"
"notify"
"publish"
"subscribe"
"unpublish"
"unsubscribe"
"net.espeak.services.events.intf.ESListenerIntf"
"net.espeak.services.events.intf.DistributorIntf"
"net.espeak.jesi.event.coredist.ESCoreDistributorIntf"
"net.espeak.infra.cci.events.Event"
"service.create"
"service.delete"
"service.mutate"
"service.access"
"service.pause"
"service.resume"
"service.panic"
"service.genericInfo"
"management.service.create"
"management.service.coldreset"
```


09730E-10600

Publication of Core-generated Events

The e-speak core sends events to the core distributor as a Protocol Data Unit containing a MessageForResource (xref to ESPDU section in communications chapter). The payload field of the MessageForResource is the event. The payloadType of MessageForResource is set to EVENT.

The e-speak Core does not subscribe to the Core Distributor (as an ordinary Client would).

Distributor Vocabulary

A vocabulary is defined in which Distributors can be registered.

Table 11 Distributor vocabulary

Attribute Name	Value Type	Comment	Meaning
Name	String	Default value "BaseDistributorVocabulary"	
Type	String		
ESGroup	String		
ContractNames	String		
ServiceName	String		Name assigned to Distributor
ServiceType	String		Type of distributor
EventTypes	String	Multivalued	Event types handled
Persistent	Boolean	always false	True if Distributor state survives Core restart
Buffered	Boolean	always false	True if Distributor is able to accept events faster than it can forward them
Secured	Boolean	always false	True if event state is tamper proof
QOSLevel	Integer	always 0	Quality of service level assigned by Distributor
Multiplexed	String	Multivalued	Type of aggregation and summarization

The Service Name, Service Type, and Event Types are strings that are assumed to have meaning to Publishers and Subscribers who have discovered the Distributor. For example, the Core Distributor could be described with a Service Name of "Core", a Service Type of "Core", and Event Types of "Repository" and "Metadata".

The Persistent, Buffered, Secure, and QOSLevel attributes must be set as shown because the current Distributor implementation does not support these features. The Multiplexed attribute can be set by Distributors to describe how they combine events. For example, a Distributor can aggregate billing events from a particular customer and publish an aggregate event to the subscribers. The values assigned are assumed to have meaning to the Publishers and Subscribers of the events.

Events in a Distributed Environment (Informational)

Events are messages that trigger special actions by the recipients. In particular, when a Client receives an Event, the callback registered for this Event is invoked. It would be inappropriate for the Remote Resource Handler to invoke the callback. In fact, the Remote Resource Handler has no idea what to do with the Event. As currently implemented, no special action is needed. The result is delivery of the Event to the Client with no special action on the part of the Remote Resource Handler.

The state of Resources exported by value and the metadata of all exported Resources is not synchronized by default. Clients wishing to synchronize exported or imported Resources register for the Core-generated metadata and Resource Events. They also subscribe to the Resource Event if the Core-managed Resource is being exported by value.

Care is needed to avoid cycles. Consider an exported Resource that has its metadata changed on the importing side. Assume that a Client on each Logical Machine has subscribed to metadata Events for this Resource with Core Distributors from both Logical Machines. When one Client makes a change, they both get the Event.

Other cycles can occur. Two Clients that make changes to the metadata while the Events are propagating can generate a cycle that is not broken so simply. The problem is that they are both changing the same item without synchronizing. Such conditions are almost certainly programming errors. No action taken by any e-speakcomponent can be guaranteed to break such cycles. Only the Clients have sufficient information to detect the problem.

SECRET

00802T 120800 09733027 2005760

Chapter 9 Management

Two concepts underpin the manageability of e-speak Resources and Clients.

- **Managed State:** a defined service state embodying the life cycle of a service.
- **Managed Variable Table:** sets of values that can be affected by a manager for the purposes of configuration and control.

Managed Life Cycle

The full state transition diagram is as follows.

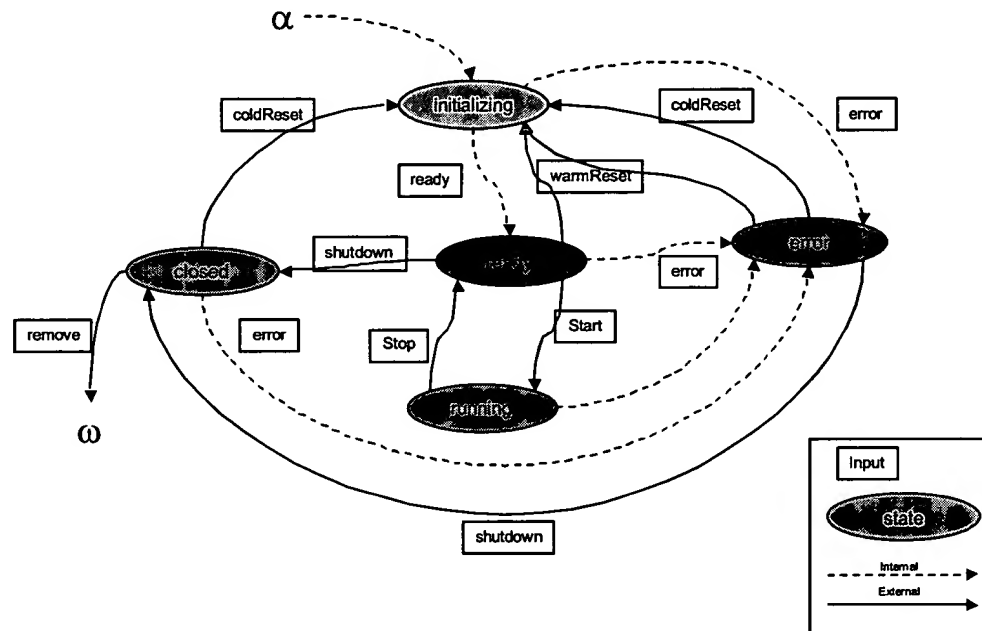


Figure 16 Managed Service Lifecycle

State Descriptions

Initializing

The internal dynamic state of the service is being constructed, for example: a policy manager is being queried for configuration information and resources are being discovered via search recipes or yellow pages servers. When the service finishes this work it moves asynchronously into the ready or error states.

Ready

The service is in a ready to run situation, this state is also equivalent to a stopped or paused state.

The service is running and responding to methods invoked on its operational interfaces. If an error occurs which implies that the service cannot continue to run it should move into the error state.

The service has some problem and is awaiting management action on what to do next.

The service has removed/deleted much of its internal state and awaits either a coldReset or remove transitions.

An input is the trigger that causes a state transition to occur. In any given state there is a defined set of permissible inputs that are available, i.e. only those that are depicted in the diagram as leaving the current state and connecting with the next state. To attempt to perform any other transition is illegal. Note that many inputs can have the same name (e.g. error) yet there is no ambiguity as long as the originating state is different.

Clients can provide any input with impunity. However, a management agent can request only provide external inputs. For example, the manager might reasonably request that a client perform a warm reset, but not to become ready; the client alone can provide this input i.e. when its internal initialization process has completed.

- **start:** move into the running state. Start to handle invocations on operational interfaces.
- **stop:** move into the ready state. Stop handling invocations on operational interfaces.
- **ready:** move into the ready state having finished initialization.

- error: move into the error state, this transition is valid from any state.
- shutdown: clean up any internal state required and move into the closed state. This transition should not cause the deregistering of resources from the repository.
- coldReset: cause a from complete reinitialization of the service and move into the initializing state. The only exemption is that resources that are already registered should not be reregistered.
- warmReset: cause a partial reinitialization of the service i.e. retaining some of the existing service state move into the initializing state.
- remove: cause the service to remove itself from existence. Any non-persistent resources should be deregistered from the repository.

Managed Variable Tables

A managed variable table is at it's simplest a table of name/string value pairs that exist within the client but to which a manager has some level of access. Thus a management agent can control those aspects of a services behavior that is affected by those variables to which it has access.

There is a degree of configurability associated with managed variables and their variables that permit something more sophisticated than the simple get and set operations one would expect to find.

Each table itself has a name to distinguish it from other tables. As we shall see later, the managed service model itself provides for two such tables.

There is a restriction on variable table usage: each name in a variable table must be unique within that table. It is not possible to implement lists by having many entries with the same name.

Configuration Parameter Table

The configuration parameter table is an instance of a managed variable table with a reserved name that identifies it as such. The table holds generic configuration data for the client.

Resource Table

The resource table is another instance of a managed variable table, identical in behavior to the configuration parameter table except that the names in the client's table refer to other services with which the client has some relationship. For example, if a particular client makes use of a mail service then this relationship can be made visible to a management agent through the resource table. Thus a management agent might reconfigure the client to use an alternative but equivalent service. While there might seem no obvious need to separate out this particular aspect of configuration, doing so makes it possible for a management agent to discover the topology and integrity of a network of connected services without the need for service specific interpretation of the variable table (all entries in the resource table are resources).

The name used for an entry in a resource table can be any symbolic name the client chooses, while the value must be the valid e-speak ESName of the actual service.

Managed Service Interface

All e-speak Resources that are manageable implement the ManagedService interface. This applies whether the Resources are external to the e-speak Core, or Core-managed.

```
interface ManagedServiceIntf{
    String getName()
    throws ESInvocationException;

    String getDescription()
    throws ESInvocationException;

    String getOwner()
```

```
throws ESInvocationException;

String getUptime()
throws ESInvocationException;

String getVersion()
throws ESInvocationException;

String getErrorCondition()
throws ESInvocationException;

String getStaticInfo()
throws ESInvocationException;

void coldReset()
throws IllegalStateTransition, ESInvocationException;

void warmReset()
throws IllegalStateTransition, ESInvocationException;

void start()
throws IllegalStateTransition, ESInvocationException;

void stop()
throws IllegalStateTransition, ESInvocationException;

void shutdown()
throws IllegalStateTransition, ESInvocationException;

void remove()
throws IllegalStateTransition, ESInvocationException;

int getState()
throws ESInvocationException;

VariableEntry[] getVariableEntries()
throws ESInvocationException;

String[] getVariableNames()
throws ESInvocationException;

VariableEntry getVariableEntry(String name)
throws ESInvocationException, NoSuchVariableName;

void setVariable(String name, String value)
throws ESInvocationException;
```

09733027.120800

The `coldReset` transition function cause the service to move into the initializing state and completely reinitialize. The exception `IllegalStateTransitionException` is thrown if the state is not in the ready, error or closed states.

The warmReset transition function cause the service to move into the initializing state and partially reinitialize. The exception `IllegalStateTransitionException` is thrown if the state is not in the ready or error states.

The start transition function cause the service to move into the running state and service client requests. The `IllegalStateTransitionException` exception is thrown if the state is not in the ready state.

The stop transition function cause the service to move into the ready state and stop serving client requests. The exception `IllegalStateTransitionException` is thrown if the state is not in the running state

The shutdown transition function clean up any internal state required and move into the closed state. This transition should not cause the deregistering of resources from the repository. The exception `IllegalStateTransitionException` is thrown, if the state is already in the closed state.

The remove transition function causes the service to remove itself from existence. Any non-persistent resources should be deregistered. The exception `IllegalStateTransitionException` is thrown if the state is not in the closed state.

The method `getState` return the current state: an integer value from 0 to 4.

The value returned is interpreted as follows.

- Initializing(0) - the service is constructing its internal data structures and finding other services which is needs to function.
- Ready(1) - the service is fully constructed and ready to run.
- Running(2) - the service is running and handling methods on its operational interfaces.
- Closed(3) - the service has deleted much of its internal state and closed any open connections to files or other services.
- Error(4) - The service has encountered an error preventing the service from continuing to operate.

The Variable Table

Each manageable Resource maintains a table of name value pairs, which contains whatever information that Resource wishes to expose to the management agent. The table entries can be either read only or read write.

```
class VariableEntry {  
    String name;  
    String value;  
    int updateType;  
}
```

The method `getVariableEntries` returns the table as an array of `VariableEntry`'s. Each `VariableEntry` object contains the name, the value & update information.

The method `getVariableNames` returns an array of strings - one element in the array for each variable.

The method `getVariableEntry` returns the entry in the table for variable identified in the parameter name.

The method `setVariable` sets the variable identified by the parameter name to the string in the value parameter.

The Resource Table

The managed Resource maintains a table of name-Resource pairs. This table contains all the Resources that the element depends on i.e. uses. The table entries can be either read only or read write.

```
class ResourceEntry {  
    String name;  
    ESName resource;  
    int updateType;  
}
```

The method `getResourceEntries` returns the table as an array of `ResourceEntry`. Each entry contains a string that name for the resource, the `ESName` of the resource (URL) and the update information.

The method `getResourceNames` returns an array of strings, one element for each entry in the resource table.

The method `getResourceEntry(String name)` returns the entry in the table for the named resource.

The method `setResource` sets the Resource identified by the name parameter to the ESName supplied in the resource parameter.

003027 120300 09733027 2205260

Chapter 10 Repository (Informational)

The Repository is not part of the e-speak architecture because Clients have no direct interaction with it. However, understanding the operation of the Repository helps in understanding other parts of the architecture. Also, the behavior of the system depends on how the Repository is configured. This chapter describes the reference implementation, the Core-Repository interfaces for including Repositories of different internal structures, and various scalability issues.

Repository Overview

The Repository holds the data needed by the Core. This data includes the Resource metadata as well as the internal state of Core-managed Resources. The Repository is also read by the Lookup Service when a Client requests a lookup. These two operations have different design points. Access to metadata and Core-managed Resources is done frequently and needs to be low latency. Lookup requests are akin to database queries; they are less latency sensitive but must be completed relatively quickly.

Repository Structure

To support the conflicting goals of flexible query lookup on a large persistent set and rapid access to a smaller, transient subset, the reference implementation of the Repository described here is divided into two components: the *Repository Database* and the *Repository Access Table*.

The Repository Database provides persistent storage and efficient lookup request processing. This component is left parameterized in the Core-Repository interface. All that is needed is an appropriate database interface. This design allows different implementations of the Repository to select the most appropriate database based on relevant business and technical considerations.

A very broad range of persistent repository implementation is allowed. This Repository Database interface gives another architectural degree of freedom. For instance, in the case of a battery-backed RAM device or in situations where persistence is simply not a requirement, a pure RAM-based Repository Database implementation is feasible. Thus, the Repository Database need not have a large footprint.

The second component, the Repository Access Table, is fully resident in memory in the reference implementation. This access table is rebuilt from data in the Repository Database as part of a system restart. The access table supports a fast associative lookup of information based on Repository Handles. It can be a cache of the Repository data, or it might be large enough to hold all the data needed for Resource access.

Information Flow

Every e-speak installation comes with an in-memory Repository that does not support persistence. To add the feature of scalability, a *glue* layer must be provided to convert Core requests to the Repository into meaningful requests to the selected implementation. This glue layer must implement the information flow methods described in this section. In addition, the glue layer can also include interfaces specific to the selected Repository implementation, such as setting controls.

The Repository Database has two interfaces used by the Core. The Core-Repository interfaces have methods to:

- Register and unregister Resources
- Access the metadata corresponding to a given Repository Handle
- Modify the metadata corresponding to a given Repository Handle

- The Client can access these methods only indirectly by invoking methods in the Contract, Name Frame, and MetaResource. The following illustrate the methods that need to be supported in these interfaces. The exact signatures and functions vary from implementation to implementation. In the current implementation these interfaces can be found in `net.espeak.infra.core.repository.Repository`

```

public RepositoryHandle registerDescription(
String          name,
ResourceDescription d,
ResourceSpecification s)
throws InvalidSpecificationException;

public void unregisterDescription(
RepositoryHandle handle)
throws StaleHandleException;

public ResourceDescription accessDescription (
RepositoryHandle handle)
throws StaleHandleException;

public ResourceSpecification accessSpec (
RepositoryHandle handle)
throws StaleHandleException;

public RepositoryHandle mutateDescription (
RepositoryHandle      handle,
ResourceDescription d,
ResourceSpecification s)
throws StaleHandleException;

```

The second interface is presented to the Core by the Repository to invoke the Lookup Service for a Repository lookup request. This interface is invoked when the Client does a lookup in a Name Frame:

```
public RepositoryHandle[] find (SearchRecipe recipe)
throws InvalidSearchRequestException;
```

The Repository can access permanent storage, but the protocol used for such access is not part of thee-speak architecture.

The keyIndexType field: Efficient Repository Lookup

In DBMS, indexes are the primary means of reducing the volume of data that must be fetched and processed in response to a query. If there were no indexes used for resource description attributes in an e-speak repository, every resource defined against a particular vocabulary needs to be examined to see if it matches the constraints specified in the search recipe. This would cause very slow performance on lookups when large numbers of resources are registered in the e-speak Core. So there needs to be a way of specifying which attributes properties within a Vocabulary are the 'key' attributes so that some indexing scheme can be added.

It is not reasonable to index each and every attribute in a resource description. The more indexes that you have, the more overhead in registering descriptions and also the memory requirement becomes more for in-memory repository. It does not make sense to index attributes that are not going to be frequently used in constraints. Therefore, there needs to be a way of specifying which attribute properties within a vocabulary are the 'key' attributes so that some indexing scheme can be implemented on these 'key' attribute properties.

This is the purpose of the keyIndexType field in AttributeProperty (xref to core managed resource Vocabulary section). Valid values of keyIndexType are: NO_INDEX, HASH_INDEX and TREE_INDEX. If the value is HASH_INDEX or TREE_INDEX the attribute is used as an index by the DBMS.

003027 120800

Chapter 11 Localization

A key factor in global acceptance of a software package is its ability to be customized to the location running the software. It is very frustrating for a user to have to read messages in a language other than their own native language or interpret numbers using a foreign format. Imagine if you had to understand an error message written in German, or recognized that the string "06/01/99" really means January 6th and not June 1st.

This chapter describes how to support localizing e-speak for native language and locale-dependent number & date formats. This design is implemented in the current release of e-speak. However, currently all entities have the same underlying data catalog to get their localized strings.

Current Implementation

The current code has hard-coded strings for all display messages and exception details. It also hard-codes the format of number and date/time representations.

For example:

- `net.espeak.infra.core.startup.StartESCore` prints a message when the core is initialized using `System.out.print("Starting ES Core Server with Rendezvous of \"+ popURL + "\". ");` and `System.out.println("started.");`.
- `Value.getString()` simply calls the `toString` method of the data type object represented by the `Value` class.

Requirements

String Messages

There are three requirements for string messages:

- A framework implementation which supports the use of localized string messages.
- A English implementation of all string messages within the core, cci and client packages, using the framework created above.
- Additional language implementations as required by our customers.

Framework

- Any time the message text is moved away from the code that produces the message, confusion and incorrect messaging is likely. It is important, therefore, that the framework minimized the confusion and makes it difficult to issue the incorrect message. A hierarchical structure must be supported for the specification of the message to be issued.
- Messages are rarely static, i.e., they often contain concatenations of variable values in the middle of the message. The framework must support the substitution of variable values in the body of the message.
- The framework must support the specification of the location and language of the user. If support for the requested location and language are not implemented, the framework should provide the closest match to the requested location and language available.
- A likely scenario includes the core running in one locale and the client running in a different locale. The framework must support a core issuing a message in the client's locale language.
- During development phases, the framework should throw exceptions if the invocation of the messaging methods are coded incorrectly (e.g., a message id that is not valid), but in the release the framework should make a best attempt to format the message for the user.

- ## English implementation

- ## Additional language implementations

- ## Number & Date Formats

- Vocabulary attributes
- Value class string representations

Vocabulary Attributes

Three new data types should be supported which provide for locale-defined formats. They are:

- **Decimal:** This data type provides for a decimal representation of a number in a user-defined pattern. The pattern can be derived from the locale-defined format or customized by the user. For example: Decimal number = new Decimal("###,###.##");
- **Currency:** This data type provides a specialized Decimal format that includes the currency symbol and format defined by the user's locale.
- **Per:** This data type provides a specialized Decimal format for percentages using the symbols and format defined by the user's locale.

Value class string representations

The Value class getString method should return a string representation that is customized by the user's locale formats. Specifically:

Timestamp	Date
Time	Decimal
Currency	Percent
Numeric data type (Long, Double, Float, etc.)	

High-level Design

The implementation for the Number and Date formats are left to the Vocabulary team. This document only addresses the String Message requirements. Shown below is the class diagram for the classes implementing localization.

ESString
messageID:String
info:Object[]
ESString()
ESString(String)
ESString(String, Object)
ESString(String, int)
ESString(String, Object, Object)
ESString(String, Object[])
toString():String
receiveObject(MessageInputStream):Object
sendObject(MessageOutputStream):void

ESStrings
table:ESMap
contents:Object[][]
getKeys():Enumeration
handleGetObject(String):Object

ESText
myResources:ResourceBundle
myCustomResources:ESList
throwExceptions:boolean
initialize(String):void
throwExceptions():boolean
setThrowExceptions(boolean):void
getLocaleString(Object):String
getMessage(String):String
getMessage(String, Object):String
getMessage(String, Object, Object):String
getMessage(String, Object[]):String
findMessage(String):String
isDigit(String):boolean

ESText, ESStrings & ESString classes

Three new classes are defined. ESText is the retrieval class and ESStrings is the language dependent implementation class. ESString is a logical extension to String which performs the localization at the last possible moment (client in most cases).

The unitize() method needs to be called by each process that uses the ESText facility. If this method is not called, the first invocation of getMessage defaults to the e-speak base class. The unitize method uses the java.util.ResourceBundle class to discover the language defined strings. ESText supports multiple base classes. After it is unitized, it can be called multiple times with different base class name parameters. When ESText looks up a message, it searches all the supplied base classes to resolve the message ID.

The throwExceptions and setThrowException methods return and specify if exceptions are thrown for detected problems (see below).

Additional getMessage prototypes can be created with multiple params if the need arises.

The base implementation for ESStrings looks like the following:

```
public class ESStrings extends ListResourceBundle
{
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents = {
        {"net.espeak.startup.Hello", "template for ID1"},
        {"net.espeak.eslib.ESFolder.dup", "template for ID2"}
    };
}
```

Each additional language looks like the following:

```
public class ESStrings_de extends ListResourceBundle
{
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents = {
        {"net.espeak.startup.Hello",
        "German override for ID1"},
        {"net.espeak.eslib.ESFolder.dup",
        "German override for ID2"}
    };
}
```

Class names are searched in the following order:

- 1 baseclass + "_" + language + "_" + country + "_" + variant
- 2 baseclass + "_" + language + "_" + country
- 3 baseclass + "_" + language
- 4 baseclass

ResourceBundle automatically defers to this search order for any message ID that is not found in the specific language implementation or if the specific language implementation is missing.

The optional param values are substituted in the message text by the following rules:

- For each occurrence of the string "%n", the string is replaced by the object[n].toString() value. Note that this is a zero-based index.
- If "n" is out of bounds for the supplied params, the string "n/a" should be substituted. Note: during development, this situation threw an exception.
- If a param is not referenced by the message, the param should be ignored. Note: during development this situation is thrown an exception.
- Multiple references to the same param should be valid (e.g., "%0 blah %0").
- If the included object is a localizable object (Timestamp, Date, Time or Number) the locale-defined formatting rules are applied to this object.
- To code a percent sign in the message, code two percent signs (%%).
- To include all passed parameters, code "%all" in the message template. This is replaced by [arg0, arg1, ...].

The last class is ESString. This class logically extends the java.lang.String class for localization. It accepts a message ID and optional data objects in the same way as ESString does. The toString method localizes the message when it is called rather than when it is constructed.

Usage example

Below is an example of how the StartESCore message can be coded. The ESStrings.java class contains the following:

```
public class ESStrings extends ListResourceBundle
{
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents = {
        {"es.core.startup.Hello1",
        "Starting ES Core Server with " +
        "Rendezvous of \"%0\"..."},
        {"es.core.startup.Hello2", " started."}
    };
}
```

The StartESCore.java class contains the following:

```
System.out.print(ESText.getMessage(  
    "es.core.startup.Hello1", popURL));  
System.out.println(ESText.getMessage(  
    "es.core.startup.Hello2"));
```

The ESString class is used as follows:

```
throw new StaleEntryAccessException(  
    new ESString("es.core.startup.Hello2"));
```

Additional design considerations

ID specifications

To simplify the ID generation and reduce the chances of duplications, the IDs should follow the following convention:

- Use the dot format to specify the hierarchy. For example, message IDs for the StartESCore class should be "es.core.startup.StartESCore.*".
- The last node should be a short description string denoting the message. For example, the startup message issued today by StartESCore would have the ID of "es.core.starStartESCore.Hello".
- Messages are defined in the ESStrings class in sorted ID order.

Core generated exceptions

Because it is possible that the core is running in a different locale from the client, any message text produced in the core that is destined for a client should be specified in the client language. To do this, modify the exception classes to pass the additional objects instead of the text. The exception.getMessage code on the client side uses the ESText class to map the exception number to a message ID and performs the substitutions in the client's language.

Client usage

Client applications can use these classes as well. They need only call ESText.initialize() with the base class name for their ESStrings equivalent.

ESString Wire Format

```
class ESString{
    String messageID;
    Object[] info;
}
```

The message ID specifies the text of the message in either the service-defined message catalog or the e-speak defined catalog. The message ID is used to retrieve a message template from the catalog (ESStrings). The optional Objects are substituted into the message based upon the syntax of the message template.

Message templates can contain the "%" (percent sign) symbol followed by a number. The number the index info object. The percent sign (and the number following it) are substituted with the toString value of the associated object.

An example entry from the current e-speak message catalog (net.espeak.util.ESStrings)

```
MessageID: "net.espeak.exception.4"
Message template: "Parameter '%0' invalid type, expected '%1'"
```

09733027-120800

Chapter 12 Future Developments

The next release of e-speak will integrate the current e-speak Core with the Web Access architecture (see e-speak Web Access Architecture). It will also implement the localization architecture described in Chapter 11, "Localization".

00802T" 220EE460

Future Developments

09733097-120300

Glossary

This chapter needs check to make sure we are not using terms no longer needed. Terms to do with security (keys and locks) need to be removed. New terms need to be added: certificate, key, PKI, ACI, Principal, URL, ESPDU..... probably others.

Term	Meaning
Advertising Service	A service for looking up resources not registered in the local Repository. It returns zero or more Connection Objects.
Arbitration policy	A specification within the search request accessor for naming that provides the logic to resolve multiple matches found for a name search.
Attribute Vocabulary	See Vocabulary .
Base Vocabulary	A Vocabulary provided at system start-up.
Builder	An entity identified by a Remote Resource Handler that is used to construct the internal state of a Resource imported by value.
Certificate	A data structure assigning a Tag or name to a Subject. Certificates are signed using cryptographic techniques so they cannot be tampered with.

Term	Meaning
Certificate Issuer (CI))	A service issuing certificates to Subjects.
Client	Any active entity (e.g., a process, thread, service provider) that uses the e-speak infrastructure to process a request for a Resource.
Client library	The interface specification that defines the interface for e-speak programmers and system developers that will build e-speak-enabled applications.
Connection Manager	A Logical Machine's component that does the initial connection with another Logical Machine.
Contract	See Resource Contract .
Core	The active entity of a Logical Machine that mediates access to Resources registered in the local Repository.
Core Event Distributor	A Core-managed Resource whose purpose is to collect information on e-speak Events and make such information available to management tools within the infrastructures.
Core-managed Resource	A Resource with an internal state managed by the Core.
Distributor Service	A service that forwards published Events to subscribers.
Event	A message that results in the recipient invoking a registered callback.

Term	Meaning
Event filter	A subscription specification expressed as a set of attributes in a particular Vocabulary that must match those in the Event state in order for a Client to receive notification on publication of an Event.
Event state	A reference within an Event to its expressed set of attributes in a particular Vocabulary. These attributes must match the Event filter in order for the subscriber to receive notification of the Event.
Explicit Binding	An accessor that contains a Repository Handle.
Import Name Frame	A container that holds a name for each imported Resource.
Inbox	A Core-managed Resource used to hold request messages from the Core to a Client.
Issuer	An entity issuing a certificate. The Issuer is denoted in a certificate by its Public Key
Logical Machine	A Core and its Repository.
Lookup request	Resources with attributes matching the lookup request will be bound to a name in the Client's name space.
Lookup Service	The component that performs lookup requests used to find Resources that match attribute-value pairs in the Resource Description of Resources registered in the Repository.
Mailbox	Either an Outbox or an Inbox.

Term	Meaning
Mapping Object	An object binding an ESName to Resources or a Search Recipe.
Message	Means of Client-Core communication.
Metadata	Data that is not part of the Resource's implementation, but is used to describe and protect the Resource.
Name Frame	A Core-managed Resource that associates a string with a Mapping Object.
Name Search Policy	A name conflict resolution tool used by the Core to find the appropriate strings when looking up names in a Name Frame.
Outbox	The location where the Client places a message to request access to a Resource.
Pass-by value	A metadata field, which, when set to true, includes the state of the Resource in the Export Form.
Principal	The entity holding the Private Key corresponding to a given Public Key
Private Key	This is secret data. An entity demonstrates knowledge of this secret data by cryptographic techniques to authenticate itself. Private Keys must be kept secret
Private Security Environment (PSE)	A cryptographically secure store for Private Keys.

Term	Meaning
Protection Domain	The environment associated with a particular Outbox from which Resources can be accessed.
Publish	A request sent to the Distributor Service to publish Events.
Public Key	Non-secret data that is associated with a given Private Key by cryptographic techniques
Public Key Infrastructure (PKI)	A set of services and protocols that support the use of public and private key pairs by applications for security.
Repository	A passive entity in the Core that stores Resource metadata and the internal state of Core-managed Resources.
Repository entry	The metadata of a Resource as stored in the Repository and made available to the Core when a Client's requests to access Resources are processed.
Repository Handle	An index into the Repository associated with the metadata of a Resource.
Repository View	A Resource that can be used to limit the search for particular Resources in a large Resource Repository, much as a database view restricts a search within a database.
Resource Contract	A Resource denoting an agreement between the Client and the Resource Handler for use of a particular Resource. The agreement includes a provision for the Client to use an API known to the Resource Handler when making the request for the Resource.

Term	Meaning
Resource	The fundamental abstraction in e-speak. Consists of state and metadata.
Resource Description	The data specified for the Attribute field of the metadata as represented by the Client to the Core. See also Resource Specification.
Resource Factory	An entity that can build the internal state of a Resource requested by a Client.
Resource Handler	A Client responsible for responding to requests for access to one or more Resources.
Resource Specific Data	A metadata field of a Resource. Carries information about the Resource. Can be public or private to the Resource Handler.
Resource Specification	Consists of all metadata fields, except the Attributes field, as represented by the Client to the Core.
Session Layer Security Protocol (SLS)	The low level message protocol used by all e-speak Cores and Clients for remote communication.
Service Identity (ServiceID)	A field in the metadata that identifies a service or Resource
Simple Public Key Infrastructure (SPKI)	A specific variant of PKI developed within the Internet Engineering Task Force and used by e-speak.
State	Data a Resource needs to implement its abstraction.

09733027-120800

Term	Meaning
Subject	The entity to which the access right or name has been issued. In a certificate the Subject is denoted by its Public Key.
Vocabulary	A Resource that contains the set of attributes and value types for describing Resources.
Tag	The field in a certificate expressing an access right
Vocabulary Builder	A Core-managed Resource registered by the Lookup Service that is used to create new value types, attributes, and Vocabularies.
Vocabulary Translator	A reference to a mechanism that is used to provide interoperation between different Vocabularies by mapping attributes from one Vocabulary into another through a Translator Resource.

5

10

Appendix B

The E-Speak Programmer's Guide
Developer Release 3.03

09733027-120800

Chapter 1 Introduction

Buying a VCR with E-speak

Let's suppose we want to buy a new VCR.

Now, let's look at the differences we would find if we were to try to buy that VCR in a large department store and in a large shopping mall.

The Department store has a known location. It also has known products.

The products in the department store are generally of the same brand. In addition, these products are generally going to have known qualities, such as price, style and color. If we were looking for an odd color, we probably wouldn't find it. The price is fixed and even though several similar products might be found in slightly different price ranges, the ranges are limited.

These product *attributes* (price, color and style) are non varying and that is extremely helpful for the store in deciding what items to stock.

Unfortunately for the *Client* that needs to use the store's *Service*, it may not be possible to find the right combination of attributes to meet their requirements.

Next month, if we need a VCR with a different set of *attributes*, say a different brand in a different color, and we return to the same location and the store has closed, it is a huge inconvenience. Having never anticipated that the store might suddenly go 'off line', we are now forced to stop what we are doing and search for a new store.

Now let's go to the mall.

The first thing we do is to drop by the Directory Board at the entrance to *discover* which stores have *advertised* their ability to sell VCRs.

Armed with this information, we visit each store with our list of required *attributes* for the VCR we want, to see if they can match the price, feature set and style that we have in mind.

Several of these stores may meet our criteria. In that case we can further refine our decision making process by prioritizing our request. We may decide to make a list with price as the governing item.

If we return next month with a requirement for a VCR with different *attributes*, we still make our first stop at the Directory Board. If any of the stores that we shopped at on the last visit are off line or new ones have been added, it doesn't affect our process of buying a VCR.

After *discovering* the *Services* that sell VCRs, we make the rounds inquiring about the attribute list (color, price, style) that each sell, and since our list of stores is up to date, we automatically include new stores that have come on line, and do not bother wasting our time making inquiries of stores that have closed.

If a store has changed or *reconfigured* the *attributes* of their services, (say they no longer offer the VCR in a brand compatible with the rest of our equipment) we will find this out during our initial inquiry and they will be filtered out from our final decision making process.

Doing this discovery and inquiry process every time, may seem like an arduous task, and indeed it is when we walk around the mall and do it. Fortunately for us, in the electronic world, these are the kind of repetitive jobs computers love to do.

As you have probably guessed by now, E-speak is built along the lines of the shopping mall paradigm.

It's job, like that of a mall, is to connect *Services* with *Clients*.

Services may come and go, so they need to be Dynamically Discoverable and E-speak provides for this.

The attributes (or what a service offers) may change also, so Services need to be Dynamically Configurable, and E-speak provides for this too.

E-speak does this by supporting a set of characteristics for its *Services* which allow *Clients* to *Discover* and use them.

Service Characteristic	Function
Description	Enables a Service to be <i>Discovered</i>
Interface	Allows Clients to communicate with Services
Mediating Access	Allows Services to coexist with other Services

Table 1 - Characteristics of E-speak Services

The way E-speak allows all this dynamic interaction is by separating the different roles played by the various members in an E-speak community.

The entities involved in defining Service ecosystems are categorized as follows

The roles and Jobs of the members of the E-speak community are:

Role	Job
<i>Service Developers</i>	Create new Services
<i>Service Standards Bodies</i>	Define new services
<i>Service Deployers</i>	Advertise the services
<i>Service Directories</i>	Provide locations where services are advertised
<i>Service administrators</i>	Monitor and Control Services

Table 2 - Roles and Characteristics of E-speak Components

Functions and Roles of E-speak Entities

- **Service developers** who create new Services
- **Service standards bodies** who define new services
- **Service deployers** who advertise the services
- **Service Directories** which are locations where services are advertise
- **Service administrators** who monitor and control services.
- **Service developers:** These are the developers of the actual Service implementation. Service developers design their Service to comply with the programmatic Service interfaces published by the standards bodies. Service developers are also able to wrap existing legacy applications so that they can be made available as services in the e-speak infrastructure.
- **Service standards bodies:** Standardization bodies define and publish standardized interfaces for different Service categories that are used by Service developers to write Services conforming to the standardized interface.

For example, a standards body may define a generic printer Service interface to contain the following invocation points: *print()*, and *status()*. Any Service provider interested in providing a printer Service writes their Service to support this interface.

In addition, the standardization body is responsible for identifying the Vocabulary used to describe a Service. The Vocabulary describes the attributes that are used to uniquely describe a Service. For example, a printer Vocabulary includes attributes such as Manufacturer, Modelname, DPI and so on (perhaps using XML) that is used by service deployers to advertise this Service, and queried by Clients to discover it.

- **Service deployers:** Service deployers are responsible for deploying the Services created by the Service developers. They are responsible for advertising the Services in the appropriate Vocabulary and also for handling requests to the Service.
- **Service administrators:** These administrators are responsible for monitoring, administering, and controlling the Service deployment.

- **Service Directories:** These directories are places where service providers and clients can advertise and find other services that are of interest to them.

Segregation of Services

This segregation of roles allows the Service developers to concentrate on the business logic of the Service that they are developing, the Service deployers to concentrate on advertising and handling requests to the Service, and the administrators to administer the Service. Furthermore, the existence of service directories greatly simplifies the process of building new composite services by providing a place where services can find other services. An example of such a service directory is accessible from the E-speak.hp.com site that each E-speak installation points to by default.

All the above entities get their work done by communication with the e-speak *Core*. The Core is the active entity of e-speak that acts as the mediation layer and routes messages to Services.

Figure 1 shows a typical interaction between Clients and Service providers in an e-speak community.

An e-speak community consists of a number of Cores connected to each other. Clients and Service providers join the community through these Cores.

Clients search for Services registered in the community, and when they are successful, they gain access to the Service using a Service proxy (stub) of the discovered Service.

The Clients need to know only the interface of the Service in order to invoke object-oriented functions (methods) on it.

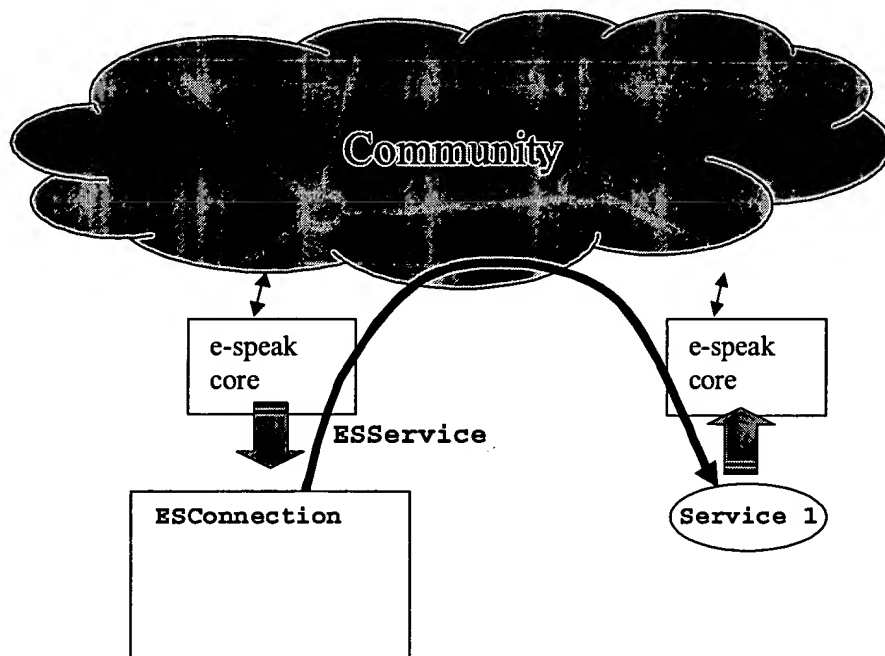


Figure 1 Typical interaction between Clients and Service providers

All access to e-speak go through the E-speak core, unlike other strategies where only discovery is done through an infrastructure and communications thereafter takes place directly between servers and clients.

The advantages of this are improved:

- Security mechanisms
- Accounting
- Auditing
- Billing.

Programming Model

The primary goal of E-speak is to simplify the development, deployment, and management of communities of Services on the Internet.

To accomplish this, E-speak provides three basic abstractions:

- Services
- Service Contracts
- Service Vocabularies

NOTE: *All three of these abstractions are first class entities in E-Speak*

E-speak Services

Essentially, Services are programs written in a programming language such as Java, C, Perl, and C++.

Figure 2 illustrates the relationship between Services, their Contracts, and their Vocabularies.

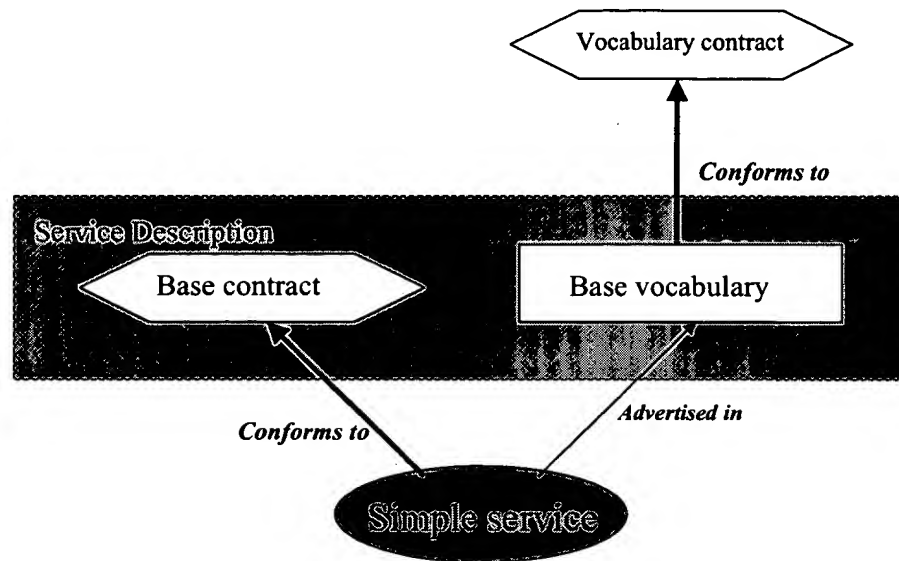


Figure 2 Relationships among Services, Contracts, and Vocabularies

NOTE: *E-speak Services conform to their Contracts because by design, they implement the interfaces outlined in their Contract.*

All services are advertised in appropriate Vocabularies.

Service Vocabularies essentially provide the scheme for constructing the description of the Service.

By decoupling Services from the Contract and the Vocabulary, E-speak allows Service implementors (the developers) to be independent of Service Deployers (the Advertisers).

This decoupling is a fundamental part of E-speak.

NOTE: *Clients are not dependent on the Service developer and deployer to communicate the Service Contracts and Vocabularies with which these Services have been described.*

The mechanism by which this happens proceeds as follows:

- 1 The Service Deployers register their Services with a group of E-speak Cores. On registering a Service, this Service is discoverable by others in the same E-speak group of Cores.
- 2 Clients use the default E-speak finders provided or write their own finders to find Services that meet their requirements.
- 3 The access permissions of the Service determine the Services that will be visible to the Clients.
- 4 On finding Services, Clients receive a remote stub to the Service.

Service Contracts

Each E-speak Service implements a set of interfaces defined in a Service Contract.

Service Contracts are first-class entities that can be discovered and used like any other Service. All Service Contracts support a base set of interfaces that provide mechanisms to create and query them.

For this reason, Contracts also can be advertised in a well-known Vocabulary—typically by a Standardization Body.

This ability to standardize Vocabularies into allows finding Services and communicating with the to proceed smoothly in E-speak, especially give the highly dynamic nature of clients and services on the web. This is the reason E-speak was designed the way it was

00000000000000000000000000000000

- Name of the attribute
- Type of values allowed

For example, the Vocabulary used to define a printer with attributes such as manufacturer, model name, DPI, speed, color, and cost is quite different from the Vocabulary that defines apparel merchandise that has attributes such as size, color, material, and cost.

The attribute color in the Printer Vocabulary may be a boolean expression that indicates whether the printer supports color printing; however, the attribute color in the Apparel Vocabulary may indicate the actual color of the piece of clothing.

This Base Vocabulary defines a set of attributes that are generic and can be used by generic Services in markets that do not have well-defined Vocabularies. All Services registered as a Vocabulary conform to the Vocabulary Contract that defines the operations that can legally be performed on Vocabularies.

In addition to the base abstractions of Vocabulary, Contracts, and Services, E-Speak supports a set of base and extended services that makes the job of deploying and using Internet-wide Services simple.

Basic Services

These are the basic set of Services that are required to get connected to the e-speak infrastructure and to create and find new user-defined Services. These Services include the following:

- **Connection**—A connection Service is used to connect and disconnect from the e-speak infrastructure.
- **Vocabulary**—A Vocabulary Service allows the creation of new Vocabularies and queries the properties of existing Vocabularies.
- **Contract**—The Contract Service is used to create and use Contracts.
- **Elements and Finders**—Service elements are used to register Services with the Core, while Service finders are used to find Services.

Extended Services

- **Events**—This Service defines a distributed Event Publisher-Subscriber model for Events. Events are distributed by a Distributor across any number of connected Cores.
- **Community**—This Service enables discovery of Services across multiple e-speak cores. Communities are Client-side abstractions of collections of groups that form the search domain. The Community is a collection of member core groups identified by group names.
- **Folders**—The Folder Service allows users to manage their Services (discovered or created) similar to how they manage local files in a standard operating system. Persistent folders appear on reconnecting and act very much like organized hierarchically persistent bookmarks.

Client-Service Interaction

As described earlier, Clients find a Service using attributes described in the Service Vocabulary. The return value of the `find()` operation is used to further connect to the Service

When Clients discover a Service, they have to specify the interface they want to use.

The Service provider's abstraction of the Service is a Service element as represented by the class `ESServiceElement`.

The Service element has, among other things, the description of the Service, the description accessor of the Service to mutate the description, and the actual implementation of the Service. It also has information about the handler of the Service, including the queue on which messages to this Service will be sent and the number of threads servicing requests to this Service.

For example, if a Service provider wants to make an existing, stand-alone application (such as `PrinterServiceImpl`), an e-speak Service, performs the following actions:

- 1 Define or modify the interface that describes the Service interface.
- 2 Create a new `ESServiceElement`.
- 3 The Service deployer provides the description of the Service in a Vocabulary, along with the implementation of the Service such as the object `PrinterServiceImpl` to the Service element. The element also contains the description accessor of the Service that can be used by the deployer to query or modify the description of the Service.
- 4 After performing these actions, the deployer can use the Service element to start the Service. The deployer can also control the concurrence of the Service by changing the number of threads that process the requests for this Service. The message queue and threads are controlled by a Service handler.

Client - Service Interaction

The following section provides a walkthrough of the interaction between and E-Speak Service and a Client that wants to use it.

First a Service must exist for the Client to be able to find it.

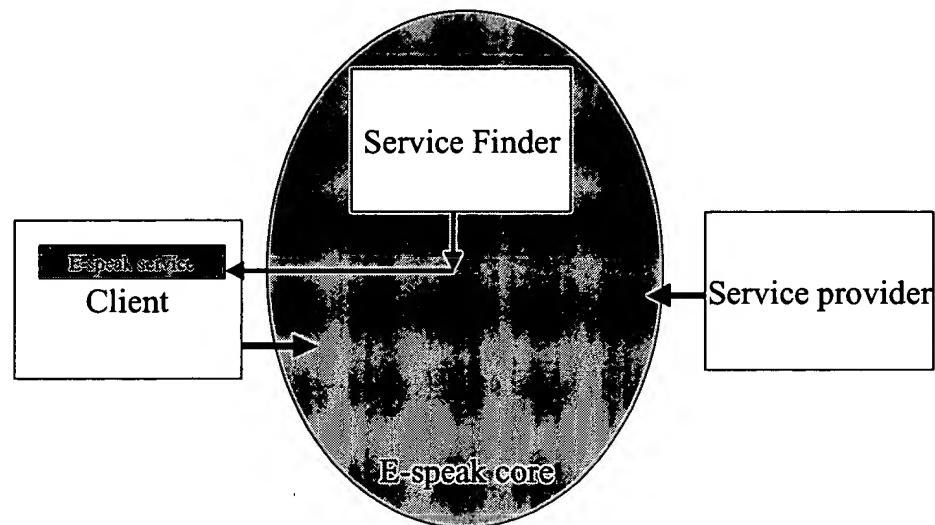


Figure 3 Client and provider Core relationship

Client Service Usage

Clients interact with the Service with the set of interfaces which are available in the Client address space.

NOTE: *When a Client invokes an operation, a well-defined e-speak custom serialization is used to ship the invocation to the target Service through the mediating e-speak infrastructure.*

Figure 4 shows that the e-speak Core is located between the Client and the Service provider. In general, many Cores may lie between the Client's Core and the Core where the Service provider is registered.

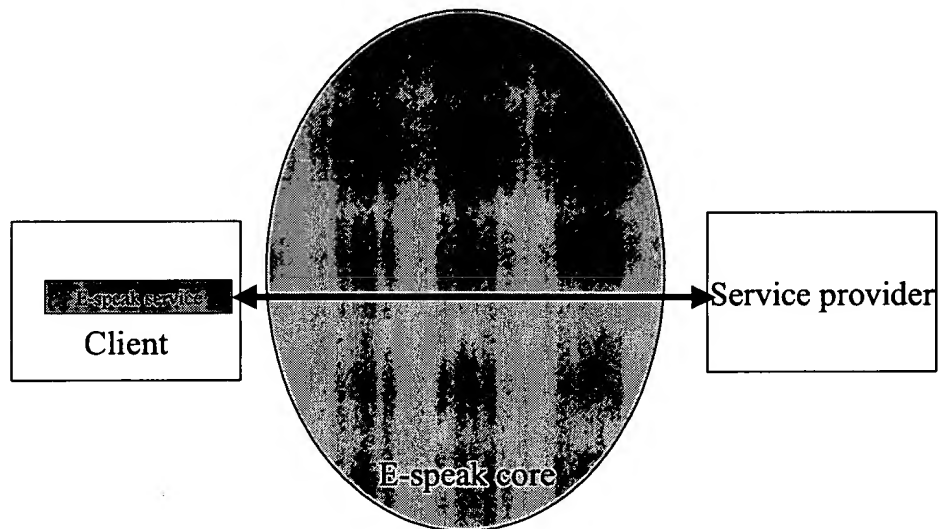


Figure 4 The e-speak Core

Resource Descriptions

E-speak makes a distinction between the data representing the state of a Resource and the data describing the management of the Resource. The Core mediates access to any registered Resource.

NOTE: *E-speak is concerned only with the Resource state of Core-managed Resources, not with the Resource state of non-Core-managed Resources.*

A Resource is described to E-speak by its metadata. The metadata is composed of a *Resource Specification* and a *Resource Description*. The Resource Description consists of information that provides the means of discovery for Clients. The Resource Specification includes:

- An Inbox that can be connected to the Resource Handler responsible for managing the Resource
- A specification of the security restrictions
- A variety of control fields

A Client registers a Resource by sending a message to a *Resource Factory* containing a Resource Description and a Resource Specification.

Together, Resource Descriptions and Resource Specifications include all information the Core needs to enforce the policies specified by the Client registering the Resource. If the registration succeeds, the Core returns a name bound to this Resource to the Inbox specified by the Callback Resource in the Outbox envelope.

E-Speak Programming Interfaces

There are two interface options available with E-speak:

- Java
- XML

JESI, based on Java, allows programmers to interact with the e-speak core or services through APIs (Application Programming Interfaces)

Web Access, based on XML, enables users to interact with the E-speak core or services through standard web browsers, by returning HTML or XML documents in HTML or XML

JESI

Jesi provides the interface for e-speak to environments that use programmatic environments such as Java.

- In J-ESI, the connection between Clients and the E-speak infrastructure is represented by an **ESConnection**.
- When a connection is established between a Client and the E-speak Core, the capabilities granted to operations performed on this connection depend on the credentials that are presented.
- **ESConnection** supports APIs that return the Base Vocabulary and Contract Services.

After a connection is established, the Client receives a 'stub' that is then used to communicate with the Server.

All communications still goes through the E-speak infrastructure to insure security, accounting and mitigation.

Some of the more important classes are:

- **ESContact**
- **EsVocabulary**

Some of the more important methods deal with finders:

- Vocabulary finders
- Contract finders
- Service finders
- Folder finders

Web Access

Web access provides the interface for E-speak to standard environments in the Internet and XML-based e-Services solutions. Of particular interest are:

- Provide access to e-speak services through standard web browsers,

- Enable e-speak services acting as services in the web (web services),
- Allowing invocation of standard, non-E-speak enabled web services from E-speak clients,
- Provide access from and to XML-based E-speak services,
- Allow e-speak services to interact based on the XML document exchange model using various transports (HTTP, TCP, VPN connections).

The architecture of Web Access is defined in the Web Access Architecture Document. Web Access internally uses “E-speak XML”¹ to represent “content”.

The term “**content**” refers here to all kinds of information related to E-speak, processed by Web Access and the E-speak core. It captures functions of the E-speak core.

Examples are search queries in order to find E-speak services, vocabulary descriptions, E-speak management information, service invocations and results passed back to requesting services and so forth.

“**Content representation**” (encoding) is different at different stages in the system, and content needs to be transformed to interface with external systems.

The connection points to external systems are referred to as “**adapters**” (inbound) and “**agents**” (outbound).

For instance, when a user requests a service discovery through a web browser, this query arrives in Web Access as a HTTP FORM POST request. This representation of the query content needs to be transformed into internal E-speak XML for further processing.

Reversely, the result represented in E-speak XML needs to be transformed into HTML as expected by the browser and sent back in the HTTP reply message. Primarily for the browser interface, content needs to be presented visually. “**Content presentation**” is a special kind of a transformation and is based on **XSL style sheet transformations**.

¹ “E-speak XML” refers to XML in accordance with the E-speak DTD/Schema definition

To find a service, a document defining the query for Services is sent to Web Access which will then return a document describing the Services which fit the query criteria.

The API Model typically assumes that the programmer has knowledge of the exact interface at programming time, usually through importing the IDL definitions at compile time to generate the stubs needed. This means that the interface must

remain immutable though the life of that version of the client. If the interface changes or is extended, the clients must be recompiled to handle or take advantage of the changes.

Changes or extensions in the document model can be discovered by the Client when it downloads the Schema. On the one hand the document model requires some additional effort in parsing the Schema and handling different formats for documents, but on the other hand this allows greater flexibility for the Client software since it is possible to handle a wider range of changes with recompiling.

00802T" 120800 093302"

0923067 10000

This document describes the E-speak ***Service Interface*** for Java programmers (J-ESI), pronounced J-Easy, that is used by Clients to program on the e-speak infrastructure. The document explains the programming model for Clients of the e-speak infrastructure. This document contains the following chapters:

- Developer Release X.03.03.00, September 2000**

Introduction

Traditionally, distributed applications have been viewed as applications that are run over multiple computers in a local domain. As a result, distributed environments have been geared toward tightly coupled applications. However, these environments do not meet the requirements of application deployment and interaction in the Internet domain.

Applications in the Internet domain are better characterized as Services. A Service is a piece of software that is not tightly coupled with Client applications. Services are dynamically discoverable and composable entities. E-speak allows building such loosely coupled, distributed services by supporting the notion of a *Service interface*, a *Service description* that enables Services to be discovered, and by *mediating access* to Services.

The e-speak infrastructure provides clear segregation of the roles played by different entities in enabling a robust Service ecosystem. The entities involved in defining Service ecosystems are categorized as follows:

- **Service standards bodies:** Standardization bodies define and publish standardized interfaces for different Service categories that are used by Service developers to write Services conforming to the standardized interface.

For example, a standards body may define a generic printer Service interface to contain the following invocation points: `print()`, and `status()`. Any Service provider interested in providing a printer Service writes their Service to support this interface.

In addition, the standardization body is responsible for identifying the Vocabulary used to describe a Service. The Vocabulary describes the attributes that are used to uniquely describe a Service. For example, a printer Vocabulary includes attributes such as Manufacturer, Modelname, DPI and so on (perhaps using XML) that is used by service deployers to advertise this Service, and queried by Clients to discover it.

- **Service developers:** These are the developers of the actual Service implementation. Service developers design their Service to comply with the programmatic Service interfaces published by the standards bodies. Service developers are also able to wrap existing legacy applications so that they can be made available as services in the e-speak infrastructure.
- **Service deployers:** Service deployers are responsible for deploying the Services created by the Service developers. They are responsible for advertising the Services in the appropriate Vocabulary and also for handling requests to the Service.
- **Service administrators:** These administrators are responsible for monitoring, administering, and controlling the Service deployment.
- **Service Directories:** These directories are places where service providers and clients can advertise and find other services that are of interest to them.

This segregation of roles allows the Service developers to concentrate on the business logic of the Service that they are developing, the Service deployers to concentrate on advertising and handling requests to the Service, and the administrators to administer the Service. Furthermore, the existence of service directories greatly simplifies the process of building new composite services by providing a place where services can find other services. An example of such a service directory is accessible from the e-speak.hp.com site that each e-speak installation points to by default.

J-ESI acts as the Java system call interface to the e-speak *Core* for all the above entities. The Core is the active entity of e-speak that acts as the mediation layer and routes messages to Services.

Figure 5 shows a typical interaction between Clients and Service providers in an e-speak community. An e-speak community consists of a number of Cores connected to each other. Clients and Service providers join the community through these Cores.

Clients search for Services registered in the community, and when they are successful, they gain access to the Service using a Service proxy (stub) of the discovered Service.

The Clients need to know only the interface of the Service in order to invoke object-oriented functions (methods) on it.

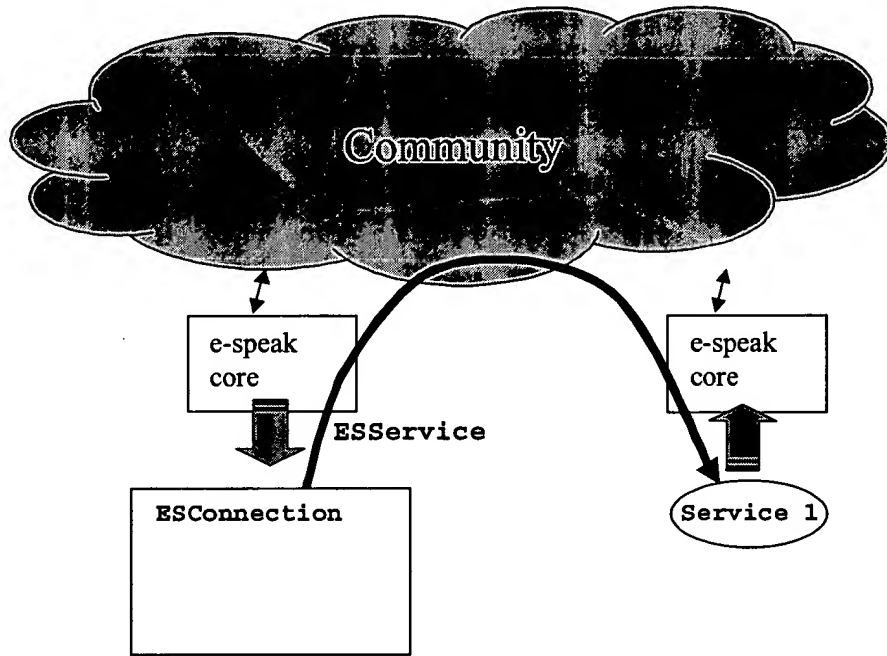


Figure 5 Typical interaction between Clients and Service providers

Unlike other distributed computing infrastructures, where the infrastructure is used only to locate the remote stub, in e-speak all subsequent accesses are mediated by the infrastructure. This mediation allows the infrastructure to enable very flexible security mechanisms that can be used to implement a wide variety of security policies. It also permits better management of the Services in the community for accounting, auditing, and billing purposes. The e-speak architectural specification has a description of the security features in e-speak. However, this release of J-ESI exposes only few security-related APIs.

Programming Model

The primary goal of e-speak is to simplify the development, deployment, and management of communities of Services on the Internet. To accomplish this, e-speak provides three basic abstractions:

- Services
- Service Contracts
- Service Vocabularies

In addition to these abstractions, e-speak provides system Services that enable Clients to write e-speak Services using J-ESI.

E-speak Services

Essentially, Services are programs written in a programming language such as Java, C, Perl, and C++. Although J-ESI makes use of some Java features, the e-speak infrastructure itself is not limited to any particular language. For example, a Service registered using J-ESI can be accessed from a Perl/Python script by using the appropriate e-speak library. Figure 6 illustrates the relationship between Services, their Contracts, and their Vocabularies.

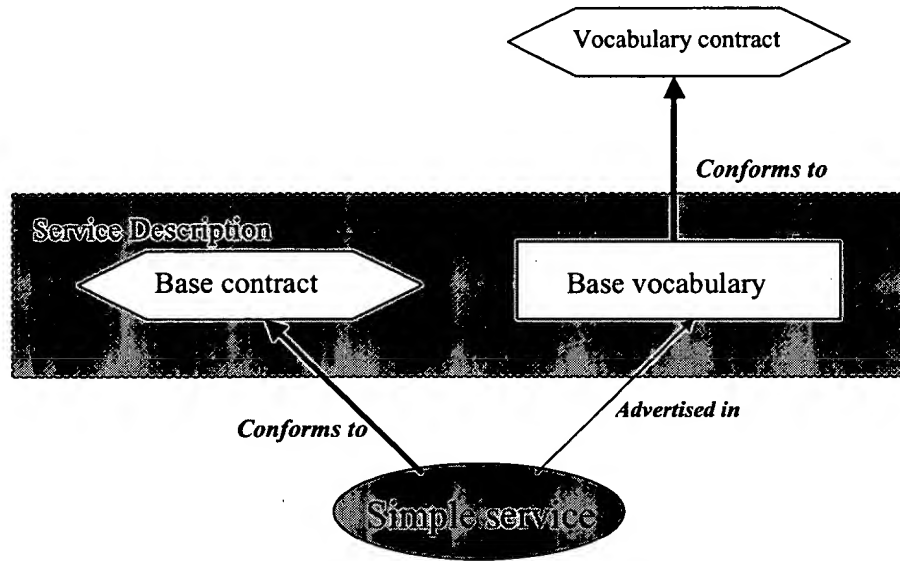


Figure 6 Relationships among Services, Contracts, and Vocabularies

Any e-speak Service conforms to its Contract because by design, it implements the interfaces that are outlined in its Contract. In addition, the Service is advertised in an appropriate Vocabulary.

Service Vocabularies essentially provide the scheme for constructing the description of the Service. All three abstractions, Services, Contracts, and Vocabularies, are first-class entities in e-speak. By decoupling Services from the Contract and the Vocabulary, e-speak allows Service implementors (who develop the actual Service implementations conforming to certain Service Contracts) to be independent of Service deployers (who simply advertise the Service in a well-known Vocabulary).

Clients are not dependent on the Service developer and deployer to communicate the Service Contracts and Vocabularies with which these Services have been described.

The Service deployers register their Services with a group of e-speak Cores. On registering a Service, this Service is discoverable by others in the same e-speak group of Cores.

Clients use the default e-speak finders that are provided with J-ESI, or write their own finders to find Services that meet their requirements. The access permissions of the Service determine the Services that will be visible to the Clients. On finding Services, Clients receive a remote stub to the Service.

Service Contracts

Each e-speak Service implements a set of interfaces defined in a Service Contract. These interfaces are often defined using the e-speak IDL, which is similar to the Java-RMI IDL. (See , "Messaging Classes", for additional information.) Service Contracts are first-class entities that can be discovered and used like any other Service. All Service Contracts support a base set of interfaces that provide mechanisms to create and query them. As a result, Contracts also can be advertised in a well-known Vocabulary—typically by a standardization body.

Service Vocabularies

A Vocabulary consists of a set of associated attributes and properties. The only properties associated with any attribute are the name of the attribute and the type of values to which these attributes are assigned. The structure of a Vocabulary is dependent on the vertical market for which it is defined.

For example, the Vocabulary used to define a printer with attributes such as manufacturer, model name, DPI, speed, color, and cost is quite different from the Vocabulary that defines apparel merchandise that has attributes such as size, color, material, and cost.

Though some of the attribute names may be the same across Vocabularies, they may have different semantics. The attribute color in the Printer Vocabulary may be a boolean expression that indicates whether the printer supports color printing; however, the attribute color in the Apparel Vocabulary may indicate the actual color of the piece of clothing.

Because Vocabularies are first-class entities, Clients and Services alike can find these Vocabularies registered in the community and subsequently make use of them. New Vocabularies are typically advertised using the Base Vocabulary that is available in the e-speak infrastructure.

This Base Vocabulary defines a set of attributes that are generic and can be used by generic Services in markets that do not have well-defined Vocabularies. All Services registered as a Vocabulary conform to the Vocabulary Contract that defines the operations that can legally be performed on Vocabularies.

E-speak System Services

In addition to the base abstractions of Vocabulary, Contracts, and Services, J-ESI supports a set of base and extended services that makes the job of deploying and using Internet-wide Services simple.

Basic Services

These are the basic set of Services that are required to get connected to the e-speak infrastructure and to create and find new user-defined Services. These Services include the following:

- **Connection**—A connection Service is used to connect and disconnect from the e-speak infrastructure.
- **Vocabulary**—A Vocabulary Service allows the creation of new Vocabularies and queries the properties of existing Vocabularies.
- **Contract**—The Contract Service is used to create and use Contracts.
- **Elements and Finders**—Service elements are used to register Services with the Core, while Service finders are used to find Services.

Extended Services

- **Events**—This Service defines a distributed Event Publisher-Subscriber model for Events. Events are distributed by a Distributor across any number of connected Cores.
- **Community**—This Service enables discovery of Services across multiple e-speak cores. Communities are Client-side abstractions of collections of groups that form the search domain. The Community is a collection of member core groups identified by group names.
- **Folders**—The Folder Service allows users to manage their Services (discovered or created) similar to how they manage local files in a standard operating system. Persistent folders appear on reconnecting and act very much like organized hierarchically persistent bookmarks.

Client-Service Interaction

As described earlier, Clients find a Service using attributes described in the Service Vocabulary. The return value of the `find()` call is a Service stub that extends `ESService` and implements the operational interface specified in the find.

When Clients discover a Service, they have to specify the interface they want to use. The stub is the Client-side abstraction of a Service that implements this interface. The Service provider's abstraction of the Service is a Service element as represented by the class `ESServiceElement`.

The Service element has, among other things, the description of the Service, the description accessor of the Service to mutate the description, and the actual implementation of the Service. It also has information about the handler of the Service, including the queue on which messages to this Service will be sent and the number of threads servicing requests to this Service.

For example, if a Service provider wants to make an existing, stand-alone application (such as `PrinterServiceImpl`), an e-speak Service, performs the following actions:

- 1 Define or modify the interface that describes the Service interface to conform to e-speak IDL.

- Clients, on the other hand, create instances of a stub to the Service when they do a find by using the e-speak finder service. They typically refer only to the interface that the stub implements and invoke methods in that interface. Clients could also choose to use the messaging APIs to communicate with Services.

The following section provides a simple example of an e-speak Service written using J-ESI that illustrates some of the basic ideas in the e-speak infrastructure.

A Client first creates a new connection to an e-speak Core. After connecting to the Core, the Client can look up or register Services. The Client locates a Service that satisfies a constraint expressed with attributes in the default Vocabulary. The result of the find Service is a stub (or proxy) to the Service provider's Service. Clients can use this stub as a network object reference and directly invoke methods on the Service (see Figure 7).

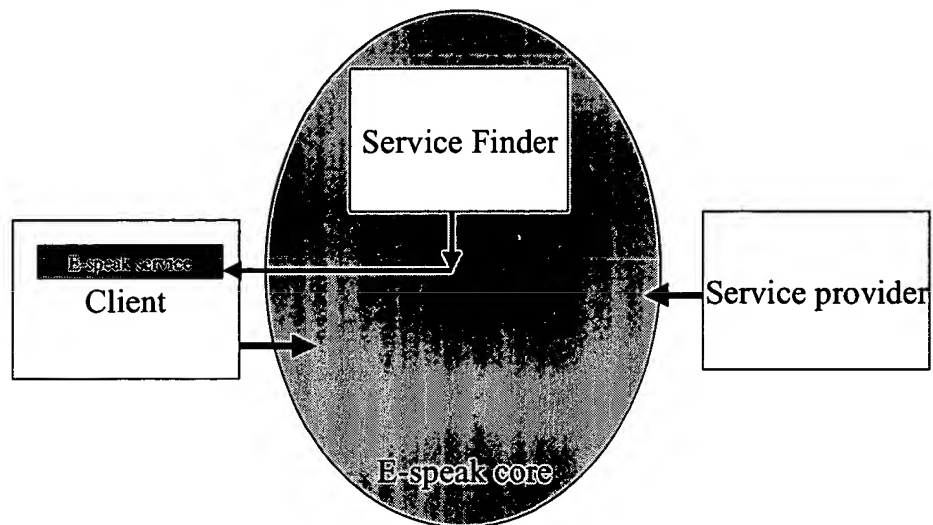


Figure 7 Client and provider Core relationship

Client Service Usage

Clients interact with the Service with the set of interfaces for which stubs are available in the Client address space. Clients can preinstall the Service stubs that are generated using the e-speak IDL stub generator, or they may acquire the stub class from the Service provider by other means. From the Client's perspective, the nature of the Service provider is insignificant beyond the requirement that it implement the interface and that its attributes satisfy the query that the Client made.

When a Client invokes an operation, a well-defined e-speak custom serialization is used to ship the invocation to the target Service through the mediating e-speak infrastructure. In so doing, all method invocations are effectively mediated. This remote invocation will work across languages and platforms, because the basic architecture does not depend on Java. The following code fragment illustrates a very simple find-and-use scenario. The Client finds a Service whose name is `printer` and invokes the `print` method on it.

```
ESConnection coreConnection = new ESConnection();
String intfName = PrinterServiceIntf.class.getName();
ESServiceFinder printFinder =
    new ESServiceFinder(coreConnection, intfName);
ESQuery printQuery = new ESQuery("Name == 'printer'");
PrinterServiceIntf printer =
    (PrinterServiceIntf) printFinder.find(printQuery);
printer.print(document);
```

Figure 8 shows that the e-speak Core is located between the Client and the Service provider. In general, many Cores may lie between the Client's Core and the Core where the Service provider is registered.

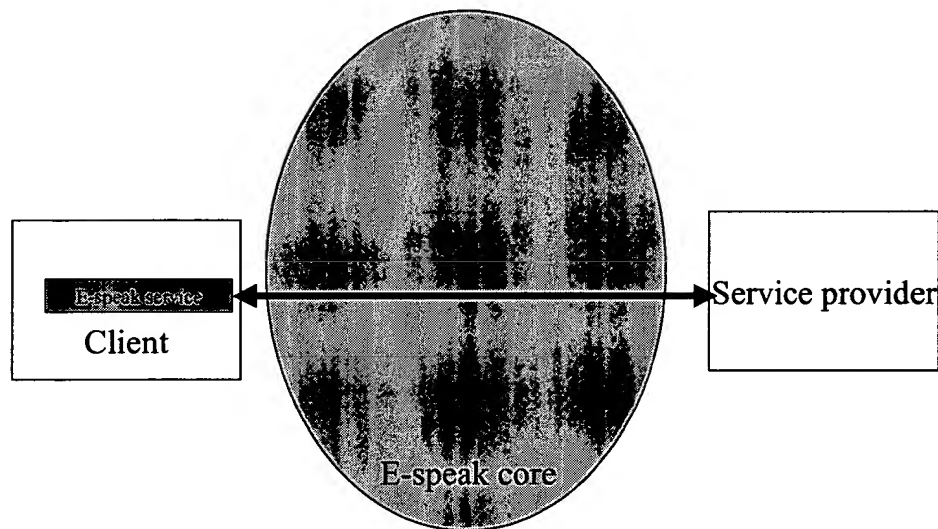


Figure 8 The e-speak Core

Service Creation

A Service provider is primarily interested in implementing a Service interface to comply with the Contract and in advertising the Service in an appropriate Vocabulary. The Service provider first connects to the e-speak Core. The provider then creates a Service element with a description of any Service Vocabularies and Contracts.

The `register()` method of `ESServiceElement` is used to register this Service with the e-speak Core. The Service element object also supports certain management APIs that allow the Service provider to start, stop, and query the status of the Services encapsulated in this Service element:

```
ESConnection coreConnection = new ESConnection();
ESServiceDescription printDescription = new
ESServiceDescription();
printDescription.addAttribute("Name", "printer");
ESServiceElement printElement =
    new ESServiceElement(coreConnection, printDescription);
PrinterServiceImpl printerImpl = new PrinterServiceImpl();
printElement.setImplementation(printerImpl);
ESAccessor printAccessor = printElement.register();
printElement.advertise();
printElement.start();
```

This code fragment creates a simple Service called `printer` that is implemented by a class called `PrinterServiceImpl`. The only attribute used in this example is the name of the Service that is set using the `addAttribute` method of `ESServiceDescription`. Clients can create more complex descriptions using custom Vocabularies, which are described in later chapters.

Once the `start` method in the Service element has been called, the `printer` Service is available for other Clients in the community to access. The actual implementation of the Service is independent of the e-speak infrastructure. In fact, the implementation can very well be legacy code potentially written in any language.

The `advertise` call in the code causes the description of the service to be advertised in the advertising directories that have been set up. By default, J-ESI advertises the description of the service to the e-services village service directory. Note that the advertising service has to be started separately in order for the `advertise` call to function appropriately.

Complete Example

This section puts together the basic ideas of e-speak into a simple end-to-end example that illustrates the sequence of steps that have to be taken on the Service provider and Client side to provide and access Services.

Typically, users have access to the IDL that defines the interface that the Service implements. In this example, the Clients and Service providers are assumed to have installed the stubs by virtue of having access to the IDL file as well as the IDL compiler. Although dynamic loading of the stubs is not directly supported, it can be implemented in user applications using standard Java class loaders, and will also be supported in the next J-ESI release that incorporates the e-speak security features.

The e-speak IDL is similar to the Java-RMI IDL. Therefore, the contents of the IDL files look similar to Java interface files. These IDL files must have a .esidl extension for the IDL compiler to recognize them as e-speak IDL files. The IDL specifications are found in , "Messaging Classes".

PrinterServiceIntf.esidl

The following is a sample esidl file that defines the interface to a printer Service that has two methods: status and print.

```
public interface PrinterServiceIntf {  
    public String status()  
    public void print(String text)  
}
```

The IDL compiler generates the following files, all of which are used by J-ESI:

```
PrinterServiceIntf.java,  
PrinterServiceStub.java, and  
PrinterServiceMessageRegistry.java
```

Except for some minor changes, such as marking the generated interface as an e-speak interface, the Service PrinterServiceIntf.java is a copy of PrinterService.esidl.

PrinterServiceStub.java is the stub class that the finder returns to the Client when it finds a Printer Service.

For every method defined in the interface, the stub class contains code to create a message, marshal parameters, and send it to the Service provider. `PrinterServiceMessageRegistry.java` does not need to be used by the Client directly, but is used by J-ESI.

PrinterServiceImpl.Java

When the IDL file is passed through the IDL compiler for Java, it produces an equivalent Java interface file. The Service implementor implements this Java interface. Therefore, the Service implementor's class (using `PrinterServiceImpl` as an example) looks as follows:

```
public class PrinterServiceImpl implements PrinterServiceIntf
{
    public String status()
        throws ESInvocationException
    {
        // Implementation to return the printer status
    }

    public void print (String text)
        throws ESInvocationException
    {
        // Implementation to print the document sent by user
    }
}
```

The Service deployer advertises the Service and handles requests to the Service. The following code fragment is written by the Service deployer:

```
public class PrintServer
{
    public static void main(String [] args)
    {
        try
        {
            ESConnection coreConnection = new
            ESConnection("file.pr");
            ESServiceDescription printDescription =
                new ESServiceDescription();
            printDescription.addAttribute("Name","printer");
            ESServiceElement printElement =
```

```

        new ESServiceElement(coreConnection,
printDescription);
        printElement.setImplementation(new
PrinterServiceImpl());
        printElement.register();
        printElement.start();
        System.out.println("Started printer Service ");
    }
    catch (Exception e)
    {
        // handle the exception
    }
}

```

The Client also runs the IDL compiler to generate the interface and stub files, and makes use of the interface in the program. For instance, a Client discovers and uses the printer as follows:

```

public class PrintClient
{
    public static void main(String [] args)
    {
        try
        {
            ESConnection coreConnection = new
ESConnection("file.pr");
            String intfName = PrinterServiceIntf.class.getName();
            ESServiceFinder printFinder =
                new ESServiceFinder(coreConnection, intfName);
            ESQuery printQuery =
                new ESQuery("Name == 'printer'");
            PrinterServiceIntf printer = (PrinterServiceIntf)
                printFinder.find(printQuery);
            String document = getDocument();
            printer.print(document);
            System.out.println(printer.status());
        }
        catch (Exception e)
        {
            // handle the exception
        }
    }
}

```

[illegible]

ESConnection supports APIs that return the Base Vocabulary and Contract Services. In addition, there are other APIs for getting management-related information that identifies users and the Core. The **ESConnection** also encapsulates the context of the actions that are performed by the Client.

The property file parameter passed to the `ESConnection` constructor provides values for the properties that are required to connect to the e-speak infrastructure. For example, the property file has entries for the following properties:

- This property specifies the name of the user who is connecting to the e-speak Core. If the property file does not provide a value for this, the default value that is used is `quest`.

- This is the password of the user. The password along with the username form the user credential.**

- esurl

```
- setTimeout(int value);
- setAsyncSendTimeout(int value);
```

- setAsyncRecvTimeout(int value);
- setFinderTimeout(int value);
- eventcontrol

This property specifies the default event control for the services being registered. If the eventcontrol property is "0" then the core will generate an event when the service metadata is mutated. If it is "1" then the core will not generate events for service mutation.

Connection Configuration

Typically, Clients create an instance of an ESConnection as follows:

```
ESConnection coreConnection = new ESConnection();
```

This API connects to the core running on default hostname, which is the localhost, and default port, which is 12346. This API is deprecated and using ESConnection constructor with a property file is recommended. In this case, a transient guest account is created and as soon as the connection is closed, this instance of the guest account is removed. Alternatively, the client can connect to the Core using

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
```

When the property file is used for creating a connection and the account name property in the property file is used, the following account creation rules are followed:

- Where the account name property is not specified, a transient guest account is created. However, if the account name is provided and no previous account of this name exists, a new account with the user and password is created.
- Where the account already exists and the user credentials are validated, the connection returns a connection with the last saved state associated with the same user credential.

After having connected to the Core, the Client can retrieve the parameters of the connection by entering

```
ESConfiguration config = coreConnection.getConfiguration();
```

The following methods in the ESConfiguration class allow the Client to retrieve the various parameters of the connection:

```
public class ESConfiguration
{
    public ESUserCredential getUserCredential();
    public String getHostName();
    public int getPortNumber();
    public String getProtocol();
    public String getURL();
    public void setHostName(String hostName);
    public void setPortNumber(int portNumber);
    public void setProtocol(String protocol);
    public String getConnectionName();
    public void setConnectionName(String name);
    public String getGroupName();
    public void setURL(String url);
    public int getCallTimeout();
    public int setCallTimeout( int newVal );
    // .. refer to javadoc for other methods
}
```

E-speak Services

There are two abstractions for a Service:

- **Service providers**—Service elements (ESServiceElement) that are used by Service providers to deploy and manage a Service.
- **Clients**—Service stubs (ESService) that are used by Clients to access an e-speak Service.

Service providers take the following steps to create a Service:

- 1 The Service developer provides implementation of the Service interface. These interfaces are encapsulated in the Contract to which the Service conforms. The actual implementation need not contain any e-speak-specific code.
- 2 The Service deployer then describes the Service in a chosen Vocabulary by setting the attribute values appropriately. The `ESServiceDescription` class is used to set these attribute values.
- 3 The Service deployer and/or administrator then creates a new `ESServiceElement` that registers and starts the Service so that other Clients of the e-speak infrastructure can find and use the Service. (See Figure 9.)

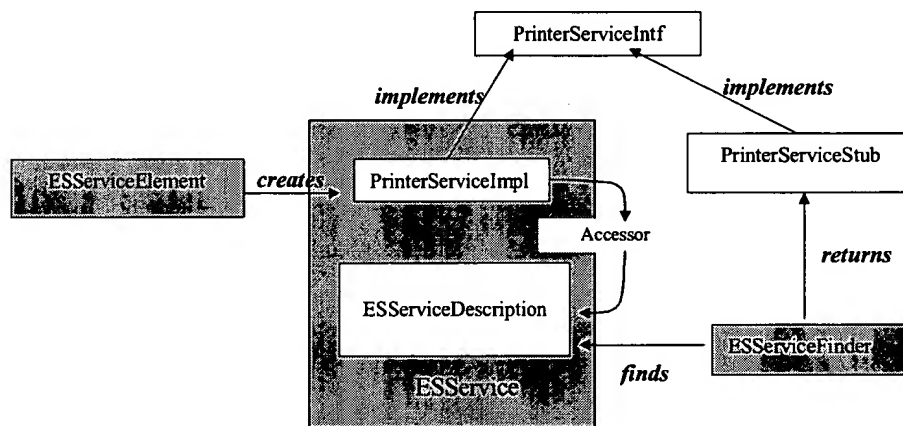


Figure 9 Creating an `ESServiceElement`

Clients typically take the following steps to find and use a Service:

- 1 Clients find Services using a Service finder that accepts queries in the Vocabulary that was used to advertise the Service. Clients also specify the interface that they expect to use to invoke the Service.
- 2 On finding a match, the Clients get a handle to a Service stub that can be used to invoke the methods that are implemented by the Service provider (see Figure 9).

The Client or Service deployer can choose to examine or modify attribute values. This is accomplished using the `ESAccessor` class.

All the three basic entities (Vocabularies, Contracts, and other Services) in e-speak can be created or found in the same manner.

The main difference between generic Services and Contracts or Vocabularies is that the interface for Vocabularies and Contracts is defined by the e-speak Core. The Core is the handler for any Vocabulary or Contract method.

Generic Services on the other hand, are not handled by the Core, and can have arbitrary interfaces that meet the requirements of the specific Service. The rest of this section describes the well-defined interfaces supported by Contracts and Vocabularies in the e-speak Core.

ESContract

Just as a Vocabulary determines the scheme for describing the attributes of Services, the Contract determines the type of Service as represented by the interfaces that it implements. Just as the Core is the handler for Vocabularies, the Core is the handler for Contracts as well.

The following method returns the name of the Contract; the name of the interface that this Contract encapsulates:

```
public String getInterfaceName();
```

The following method returns the IDL string associated with the Contract:

```
public byte [] getInterfaceDefinition();
```

In addition, the `ESContract` has methods that return the conversations that the service supports, the terms and conditions of use, and the license policy. The following entry points in `ESContract` are the relevant entry points.

```
public String getConversationScheme();  
public String getTermsOfUse();  
public String getLicense();
```

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESContract baseContract = coreConnection.getBaseContract();
```

After a Vocabulary has been defined, Clients of the Vocabulary can only query its contents to determine what properties this Vocabulary defines. The Clients cannot change the properties of the Vocabulary. Furthermore, they can query the core for all the services that are registered in this vocabulary. There are two main APIs in ESVocabulary:

The conversation scheme, terms of use, and licensing policy are not available for the base contract.

The base Vocabulary can be retrieved from an `ESConnection` using the `getBaseVocabulary()` method. The following code fragment shows how Clients get the Base Vocabulary:


```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESVocabulary baseVocab = coreConnection.getBaseVocabulary();
```

The following table shows some of the important properties that are defined in the Base Vocabulary. No other vocabulary should define and use 'Name', 'Type' and 'ESGroup' as these attributes are reserved and are used by the J-ESI library internally.

Table 3 Base Vocabulary properties

Attribute name	Type	Description
Name	String	Name of the Service
Type	String	Type of Service
ResourceSubType	String	Subtype of Service
Description	String	Description of the Service
Version	String	Version of the Service
ESDate	Date	Date associated with the Service
ESGroup	String	Group associated with the Service
ESTimestamp	Timestamp	Timestamp associated with the Service
ESCategory	String	Stringified list of categories that any service is advertised in.

Client: Finding Services

Clients to the e-speak Core typically find Services that match certain constraints. The kinds of finders are visible to the Client are:

- Vocabulary finders
- Contract finders
- Service finders
- Folder finders
- View finders

You can extend these finders to suit your needs or to use the APIs supported in these classes directly.

Setting Search Level in Finders

The following search level operations are possible in the finders. These methods are present in the abstract class `ESAbstractFinder` and are visible to all the derived finder classes such as `ESServiceFinder` and `ESContractFinder`:

```
public void setMaxToFind(int maxToFind)
public int getMaxToFind()
```

Using the above set method, it is possible to set the number of maximum services that should be returned as a result of a find call. The get method returns the current setting. The default maximum value is 1000.

```
public void setSecurityLevel(boolean flag)
public boolean getSecurityLevel()
```

Passing a true in the above set method enables stub classes to be dynamically downloaded from the service provider. Passing a false will allow classes to be loaded only from the local classpath. The getter returns the current settings for the security level. By default, the stub classes will be loaded only from the local classpath.

```
public void setSearchLevel(int flag)
public int getSearchLevel()
```

The above methods allow the user to set the level for searching the services. Passing `ESConstants.LOCAL_ONLY` will restrict the search for services to the local core repository irrespective of the community settings. `ESConstants.LOCAL_ADV`

will restrict the search to local repository and local group.
ESConstants.ADV_ONLY will result in the search being conducted in the community. The default is that the search will be conducted in LOCAL_ADV mode.

ESContractFinder

The `ESContractFinder` class is used by Clients to find Contracts. Contracts are Services that are registered with the e-speak Core just like any other Service. The Contract Finder finds only Services that have been registered as Contracts. The attribute that is used to distinguish between Contracts is the Name attribute in the Base Vocabulary. There are three important entry points in `ESContractFinder`. These are:

```
public ESContract find(ESQuery query);
public ESContract[] findAll(ESQuery query);
public ESContract [] findNext();
public void setMaxToFind(int number);
```

The find method finds a single contract that matches the constraints and preferences in the query. The findAll finds all contracts that match the query subject to the maximum number of services set in the finder.

For example, to find a Contract named `PrintContract`, use the following code fragment:

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESContractFinder printContractFinder =
    new ESContractFinder(coreConnection);
ESQuery printContractQuery =
new ESQuery("Name == 'PrintContract'");
ESContract printContract =
    printContractFinder.find(printContractQuery);
```

ESVocabularyFinder

The `ESVocabularyFinder` class is used by Clients to find Vocabularies. Finding a vocabulary is done more often than creating a vocabulary. Vocabularies are usually defined by some standard body. For example, a standard body might define a printer vocabulary which contains attributes like the speed of the printer, its location, etc. The definition of the vocabulary happens only once. It is likely that this vocabulary is registered in a community of interest. Any printer which wants to conform to this standard body finds the vocabulary and registers the printer service using this vocabulary.

Because a vocabulary is also a service, it needs to be advertised as well. Usually vocabularies are advertised in the default vocabulary. It is also possible to advertise a vocabulary in another non default vocabulary.

```
public ESVocabulary find(ESQuery query);
public ESVocabulary[] findAll(ESQuery query);
public ESVocabulary[] findNext();
public void setMaxToFind(int number);
```

For example, to find a Vocabulary named `printerVocab`, use the following code fragment:

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESVocabularyFinder printVocabFinder =
    new ESVocabularyFinder(coreConnection);
ESQuery printVocabQuery =
    new ESQuery("Name == 'printerVocab'");
ESVocabulary printVocab =
    printVocabFinder.find(printVocabQuery);
```

Instead of using `ESQuery`, you can find a vocabulary using an XML query using `ESXMLQuery`.

ESServiceFinder

The `ESServiceFinder` class is a generic finder class that is used to find user-defined Services. The interfaces here look similar to the `ESContractFinder` and `ESVocabulary` finder.

```
public ESService find(ESQuery query);
public ESService[] findAll(ESQuery query);
public ESService [] findNext();
public void setMaxTofind(int number);
```

The `ESServiceFinder` class has two constructors. One constructor which takes only `ESConnection` as parameter. `find/findAll` in this case will return the base stub (`ESService`) from which all other stubs derive. The second constructor takes `ESConnection` and the interface name (or `ESContract`) as parameters. `find/findAll` in this scenario returns the stub for the Service provider. When the interface name is specified for doing a find, the client library sends an introspection request to the potential service providers. The service providers that respond correctly to the introspection request are returned as results. This means that when the interface name is used in the find, the services that the finder returns were known to be up and ready at the time of the find. No such guarantee can be made when the finder is used without the interface name. If the Service provider is not running, the call times out if the finder timeout is set. More details on how to set the timeouts is available in the later sections.

The `ESServiceFinder` class can return multiple Services as the result of a query. However, Clients can control the maximum number of Services that are returned as the result of a find by entering the following code:

```
public void setMaxToFind(int number);
public int getMaxToFind();
```

Chapter 2 presented an example of a Client that searches for a simple printer that is described in the base Vocabulary. A slightly more complex example of a Printer that has been advertised in a non-Base Vocabulary, called a `printerVocab`, requires the following code to find the printer:

```
public class PrintClient{
    public static void main(String[] argv){
        try {
            String propertyFileName =
                new String("/users/connection.prop");
```

```

ESConnection coreConnection =
    new ESConnection(propertyFileName);
// find vocabulary
ESVocabularyFinder printVocabFinder =
    new ESVocabularyFinder(coreConnection);
ESQuery printVocabQuery =
    new ESQuery("Name == 'PrintVocab'");
ESVocabulary printVocab =
    printVocabFinder.find(printVocabQuery);
// find contract
ESContractFinder printContractFinder =
    new ESContractFinder(coreConnection);
ESQuery printContractQuery =
    new ESQuery("Name == 'PrintContract'");
ESContract printContract =
    printContractFinder.find(printContractQuery);
// find service described in vocab that conforms to
contract
    ESServiceFinder printFinder =
        new ESServiceFinder(coreConnection, printContract);
    ESQuery printQuery = new ESQuery(printVocab);
    printQuery.addConstraint("DPI == 1400");
    PrinterServiceIntf printer = (PrinterServiceIntf)
        printFinder.find(printQuery);
    String document = getDocument();
    printer.print(document);
} catch (Exception e)
{
    // handle the exception.
}
}
}

```

The finder class also supports a cursor like mechanism for clients to traverse the result set obtained as a result of performing a `findAll()` operation. The cursor mechanism is quite common in relational database APIs such as JDBC. In JDBC, the result of executing a query is a result set, and the client program can loop based on a condition that depends on whether there are additional rows in the result set. In J-ESI, on the other hand, the finder itself acts as the result set. On invoking a `findAll()` operation on the finder, the finder acts as a cursor. The client program can determine whether there are other services found by the finder by invoking the `hasMoreResults()` method on the finder. In addition, the next set of results can

be obtained from the finder by invoking the `findNext()` method. In the example below, the query essentially finds all the services that are registered in the printer vocabulary. It then proceeds to retrieve the services that are found two at a time.

```
33public static void main( String [] args ) {
    try {
        String propertyFileName = new String("/users/
connection.prop");
        ESConnection conn = new ESConnection(propertyFileName);
        ESVocabularyFinder vf = new ESVocabularyFinder(conn);
        ESVocabulary v = null ;
        try {
            v = vf.find( new ESQuery("Name == 'printerVocab'"));
        } catch( LookupFailedException ffe ) {
            ffe.printStackTrace();
            return;
        }
        ESServiceFinder finder = new ESServiceFinder(conn);
        ESQuery query = new ESQuery(v,"(Name == 'printer') or (Name !=
'printer')");
        ESService[] serviceList = null;
        int total_findall_services = 0;
        finder.setMaxToFind(2);
        serviceList = finder.findAll(query);
        total_findall_services += serviceList.length;
        while (finder.hasMoreResults()){
            serviceList = finder.findNext();
            // .. do stuff with services found in service list.
            System.out.println("number of services got: " +
                serviceList.length);
        }

        } catch( Exception ex ) {
            ex1.printStackTrace();
        }
    }
    return;
}
```

ESQuery

In general, Clients use the ESQuery class to construct queries that are either not simple name lookups in the Base Vocabulary or are constraints that are expressed in a Vocabulary that is different from the default Vocabulary. The ESQuery class has methods that allow the client to add constraints that make up the query. These constraints are string constraints whose syntax is similar to the syntax of constraint specification in OMG's OTS specification. Essentially, every constraint is parsed as if it were a disjunction of conjunctions, i.e., the constraint expression is expected to be in disjunctive normal form.

More formally, the context-free grammar that defines the syntax of the constraint strings is as follows:

```

<Expr> ::= <OrExpr>
<OrExpr> ::= <AndExpr>
           | <OrExpr> 'or' <AndExpr>
<AndExpr> ::= <NotExpr>
           | <AndExpr> 'and' <NotExpr>
<NotExpr> ::= <EqualityExpr>
           | 'not' <EqualityExpr>
<EqualityExpr> ::= <RelationalExpr>
           | <RelationalExpr> ( <OpEqual> | '!=' ) <RelationalExpr>
<RelationalExpr> ::= <InExpr>
           | <InExpr> <OpRelational> <InExpr>
<InExpr> ::= <AdditiveExpr>
           | <AdditiveExpr> 'in' <AdditiveExpr>
           | <AdditiveExpr> 'in' <ListExpr>
<AdditiveExpr> ::= <MultiplicativeExpr>
           | <AdditiveExpr> <OpAdditive> <MultiplicativeExpr>
<MultiplicativeExpr> ::= <UnaryExpr>
           | <MultiplicativeExpr> <OpMultiplicative>
<UnaryExpr>
<UnaryExpr> ::= <UnionExpr>
           | '-' <UnaryExpr>
           | 'exist' <PathExpr>
<UnionExpr> ::= <PathExpr>
           | <UnionExpr> '|' <PathExpr>
<ListExpr> ::= '[' <Arguments> ']'
<PathExpr> ::= '/'
           | '/' <RelativeLocationPath>
           | '/' <RelativeLocationPath>
           | <RelativeLocationPath>
           | <FilterExpr>

```



```

<FilterExpr> ::= <PrimaryExpr>
| <FilterExpr> '/' <RelativeLocationPath>
| <FilterExpr> '/' <RelativeLocationPath>

<RelativeLocationPath> ::= <Step>
| <RelativeLocationPath> '/' <Step>
| <RelativeLocationPath> '/' <Step>

<PrimaryExpr> ::= '$' <NCName>
| '(' <Expr> ')'
| <Literal>
| <FunctionCall>
| <Arguments>

<FunctionCall> ::= <FunctionName> '(' <Arguments> ')'
<Arguments> ::= <Expr>
| <Arguments> ',' <Expr>

<Predicate> ::= '[' <Expr> ']'
<Step> ::= <StepPredicate>

<StepPredicate> ::= <AxisSpecifier> <NodeTest>
| <StepPredicate> <Predicate>

<AxisSpecifier> ::= <AxisName> '::'
| '@'
| e

<NodeTest> ::= '*'
| <NCName> ':' '*'
| <QName>
| NodeType '(' ')'
| 'processing-instruction' '('

String_Lit ')'
<NodeType> ::= 'comment' | 'text' | 'node'
<AxisName> ::= 'ancestor' | 'ancestor-or-self' | 'attribute'
| 'child' | 'descendant' | 'descendant-or-self'
| 'following' | 'following-sibling' |
'namespace'
| 'parent' | 'preceding' | 'preceding-sibling'
| 'self'

<Qname> ::= <NCName>
| <NCName> ':' <NCName>

<Literal> ::= <Number_Lit>
| <String_Lit>
| <Boolean_Lit>
| <Date_Lit>
| <Time_Lit>
| <TimeStamp_Lit>

<OpEqual> ::= '=' | '=='

```

```

<OpRelational> ::= '<' | '<=' | '>' | '>=' | '&lt;=' | '&lt;=' | '&gt;' |
'&gt;='
<OpAdditive> ::= '+' | '-'
<OpMultiplicative> ::= <MultiplyOperator> | 'div' | 'mod'
<NCName> ::= ( <Letter> | '_' )? <NCNameChar>*
<NCNameChar> ::= <Letter> | <Digit> | '.' | '-' | '_'
| <CombiningChar> | <Extender>
<Number_Lit> ::= <Digit>* '.' <Digit>+ ( ( 'E' | 'e' ) ( '+' | '-'
' ) <Digit>+ )
<String_Lit> ::= '"' [ <AnyChar>^'"' ]* '"'
| "'" [ <AnyChar>^'"' ]* "'"
<Boolean_Lit> ::= true | false
<Date_Lit> ::= '{' ( 'd' | 'D' ) WhiteSpace* (CC | '-' ) YY? '-' MM? '-'
'DD' '}'
<Time_Lit> ::= '{' ( 't' | 'T' ) WhiteSpace* HH ':' MM ':' SS
(''
SSS S* )? ( 'Z' | '-' MM ':' SS )? '}'
<TimeStamp_Lit> ::= '{' ( 't' | 'T' ) ( 's' | 'S' ) WhiteSpace* CCYY
'-' MM '-' DD 'T' <Time_Lit>

```

Example query strings include:

```

"TestStr1 + 'String2' == 'String1String2' and TestInt - 5 < 10 and
TestDouble < 30"
"TestDate > {d 1980-08-21} and TestTimestamp <= {ts 1992-05-
12T05:33:44.111111000}",
"TestInt * 2 >= 22 and not (TestDouble >= 55.550)",
"TestInt > 10 or TestDouble == 10 or TestFloat > 10.0",
"TestBool != TRUE or TestInt -10 <= 10 and 2345.99 >= TestBigDecimal11",

```

The specialization of ESQuery called ESXMLQuery can be initialized from a file that contains a query for a Service expressed in XML:

```

public ESXMLQuery(ESConnection coreConnection,
    ESXMLFile xmlQueryFile);

```

Finding Services Using XQL

XQL can be used to find a printer described in printervocab. See, "IDL Compiler", for more details about XML DTDs used in e-speak.

The following XML page describes an XQL-based search that is passed to the ESQuery constructor:

```

<?xml version="1.0"?>

```

```

<ESpeak version="E-speak 1.0" operation="FindService">
  <resource>
    <!-- The search query-->
    <query>
      <queryBlock>
        <WHERE>
          <!-- Begin: Specify printer Vocabulary -->
          <query>
            <queryBlock>
              <WHERE>
                <condition>
                  <IN>
                    <pattern>
                      <Name>printervocab</Name>
                    </pattern>
                  </IN>
                </condition>
              </WHERE>
            </queryBlock>
          </query>
          <!-- End: Specify printer Vocabulary -->
          <!-- Begin: search conditions -->
          <condition>
            <predicate lexpr="DPI" rexpr="1400" RelOp="eq"/>
          </condition>
          <!-- End: search conditions -->
        </WHERE>
      </queryBlock>
    </query>
  </resource>
</ESpeak>

```

The Service to be discovered is specified by means of a query. An embedded query specifies the attribute Vocabulary in which attributes for the query conditions are specified. In this case, assume that this XML fragment is in a file called `printFinder.xml`.

The query searches in the Vocabulary called `printervocab` for a printer whose DPI attribute has a value of 1400.

```

<?xml version='1.0'?>
<esquery xmlns="http://www.e-speak.net/Schema/E-
speak.query.xsd">
  <result>$serviceInfo</result>

```

```

<where>
  <vocabulary name="printervocab" src="printervocab"/>
  <condition>
    printerVocab:DPI = 1400
  </condition>
</where>
<preference>
</preference>
<arbitration>
  <operator>first</operator>
  <cardinality>all</cardinality>
</arbitration>
</esquery>

```

There is also a constructor for ESXMLQuery that takes two ESXMLFiles as arguments:

```

public ESXMLQuery( ESConnection connection, ESXMLFile xmlHeader,
                  ESXMLFile xmlRequest );

```

In the example above, in the line where printQuery is constructed, this constructor is used instead of the constructor that takes in a single xml file. In addition, the contents of the xmlHeader file look as follows:

```

<?xml version='1.0'?>
<header xmlns="http://www.e-speak.net/Schema/E-
speak.header.xsd">
  <communication>
    <to>es://localhost:12346/WebAccess/FindService</to>
  </communication>
</header>

```

Queries in Multiple Vocabularies

The query can be constructed in multiple vocabularies. This feature of being able to specify queries in multiple vocabularies is similar to the notion of formulating queries in SQL that query multiple tables. Follow the steps detailed below to specify a query in multiple vocabulary.

- 1 Specify a key for each vocabulary using the following method available in ESQuery:

```

public void addVocabularyKey(String key, ESVocabulary vocab);

```

```
e.g., query.addVocabularyKey("printervocab", vocab1);
query.addVocabularyKey("defaultvocab", vocab2);
```

2 Specify the constraint in "key:attr" form instead of simple "attr" form.

```
String myConstraint = "(printerVocab:Manufacturer == 'Acme' &&
defaultvocab:Name == 'myPrinter')";
```

Constraint specifies conditions that services of interest must satisfy. The lookup service evaluates constraint against service descriptions in the repository and returns a set of matching services. Sorter is applied to the resulting set of services to order the services. Arbitration policy is used to limit the size of the resulting set if the constraint evaluation results in multiple services.

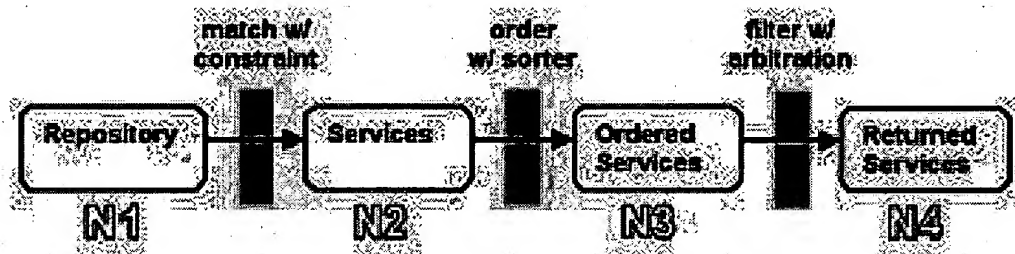


Figure 10 Order of Application for Service Sorting

The above figure illustrates the order in which constraint, sorter, and arbitration are applied to found services.

The query can be enhanced with a sorter. The specification of a sorter is similar to a notion of "order by" construct in SQL. Essentially, the user gets the finder to return the services using the order specified in the sorter parameter. This specification is done by the addSorter entry point in ESQuery.

```
public void addSorter(int type, String expression);
public void addSorter(String condition, String weight);
```

The first method takes in two arguments. The first argument should be either `ESConstants.SORTER_OPERATOR_MIN` or `ESConstants.SORTER_OPERATOR_MAX`.

```
query.addSorter(ESConstants.SORTER_OPERATOR_MIN,
"length*height*width");
```

```
query.addSorter("length < 5", "10");
query.addSorter("weight < 10:", "2");
```

```
public void setArbitrationPolicy(int policy);
```

Using Introspection

60

Another use of introspection is in the development of lightweight Clients. The Clients do not need to have any of the class files like the interface, stub files, and so forth. All these can be obtained using introspection and stub factory interfaces, and methods can be invoked using Java reflection. Thus introspection and stub factory interfaces allow building truly dynamic, lightweight, and scalable solutions.

- ESContractElement

- `ESVocabularyElement`
- `ESServiceElement`
- `ESFolderElement`
- `ESViewElement`

A similar set of classes encapsulates the descriptions of Services:

- `ESContractDescription`
- `ESVocabularyDescription`
- `ESServiceDescription`
- `ESViewDescription`
- `ESFolderDescription`

Service Description

Service descriptions are sets of attribute-value pairs expressed in a certain Vocabulary describing the Service. The Vocabulary determines the names and types of the attributes used in the description of the Service. For instance, a printer Service can be described in a Printer Vocabulary. The Printer Vocabulary in turn can contain attributes such as `DPI` that takes on an integer value, `Manufacturer`, that takes on a string value, and `Modelname` that takes on a string value. The printer itself is identified by the fact that it is manufactured by a particular manufacturer, has a particular model name, and prints at a particular DPI value.

Vocabularies, in turn, have to be described, and e-speak breaks the recursion using a Base Vocabulary that has predefined attribute properties described earlier.

All Service Descriptions are associated with two kinds of attributes:

- **Searchable attributes**—Those used in queries by Clients. These are also the attributes listed in each Vocabulary.

- **Service-specific data (Data)**—This data, which is stored along with the description of the Service, cannot be searched for, but can be accessed. This data can be as simple as String entries, or as complex as arbitrary byte code. For example, one Datum in the printer description may be the administrator's contact information (such as a phone number).

In the current API, there are two ways of specifying the descriptions:

- Using XML (This is the recommended method for specifying descriptions.)
- Using the e-speak description of objects provided in the Client library

Searchable Attributes

The attributes that can be part of any Vocabulary can be in one of the 14 e-speakbase types:

- | | |
|-------------|--------------|
| • String | • boolean |
| • byte | • char |
| • short | • int |
| • long | • float |
| • double | • BigDecimal |
| • Date | • Time |
| • Timestamp | • byte [] |

The `ESBaseDescription` class supports the addition of attributes by the `addAttribute` method. These calls take two parameters, the first is the name of the attribute, and the second parameter is the value of the parameter.

ESContractDescription

Contracts are also Services in the e-speak infrastructure. Contracts are typically described using the Base Vocabulary. Contracts encapsulate the Service IDL to which the Service is guaranteed to conform.

The `setInterfaceDefinition` methods set the IDL associated with the Contract:

```

public void setInterfaceName(String intfName);
public void setInterfaceDefinition(String idlString);
public void setInterfaceDefinition(byte[] intfClass);
public void setConversationScheme(String scheme);
public void setTermsOfUse(String terms);
public void setLicense(String license);

```

The following code segment explains how to create a contract using the contract description.

```

String propertyFileName = new String("/users/conection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);

ESContractDescription printContractDescription =
    new ESContractDescription();
printContractDescription.addAttribute("Name", "PrintContract");
printContractDescription.setInterfaceName
    ("printer.PrinterServiceIntf");
String idl = getIDLFromFile("PrinterService.esidl");
printContractDescription.setInterfaceDefinition(idl);

ESContractElement printContractElement =
    new ESContractElement(coreConnection,
    printContractDescription);
ESContract printContract = printContractElement.register();

```

ESVocabularyDescription

This class is used to describe a new Service Vocabulary that is used in descriptions of other Services. As noted earlier, the printer Service is described in the Printer Vocabulary.

There are two parts to describing a Vocabulary. First, because Vocabularies are Services that can be discovered, the attributes of the Vocabulary need to be defined so that Clients and Service providers can discover the Vocabulary. Second, the properties of the Vocabulary need to be defined so that any Service that is registered in this Vocabulary has the same properties. These properties can also be used to build queries to find Services.

Vocabularies are somewhat analogous to a table schema in relational databases. The properties that make up the Vocabulary are the columns of the table, and each Service that is registered in the Vocabulary becomes a row in the table. In keeping with the database analogy, it is assumed that the names of the properties in a Vocabulary are case insensitive.

Clients who are creating Vocabularies can set the name of the new Vocabulary they want to create using the `addAttribute()` method in the `ESBaseDescription` class.

Just as Service descriptions contain 14 basic attributes in e-speak, the Vocabulary descriptions can contain properties in those 14 types:

- String
- boolean
- byte
- char
- short
- int
- long
- float
- double
- BigDecimal
- Date
- Time
- Timestamp
- byte []

Typically, Clients make a series of calls that add various named properties. For example, to create a description of a printer Vocabulary that has three properties corresponding to the manufacturer, model name, and DPI and that indicates that a

particular property is mandatory (meaning that any Service registered in this Vocabulary has to provide a value for this property), use the following code fragment:

```
String propertyFileName = new String("/users/conection.prop");
ESConnection coreConnection = new
    ESConnection(propertyFileName);

ESVocabularyDescription printVocabDescription =
    new ESVocabularyDescription();
printVocabDescription.addAttribute("Name", "PrintVocab");
printVocabDescription.addStringProperty("Manufacturer");
printVocabDescription.addStringProperty("Modelname");
printVocabDescription.addIntegerProperty("DPI");

ESVocabularyElement printVocabElement =
    new ESVocabularyElement(coreConnection,
        printVocabDescription);
ESVocabulary printVocab = printVocabElement.register();
```

Specifying Additional Information and Essential Properties of a Vocabulary Property

The example on the previous page shows registering a simple vocabulary. It can be seen that in the example, for each property in the vocabulary, its name and type are stated. For instance, `addStringProperty("Manufacturer")` results in a vocabulary property whose name is "Manufacturer" and type is a String. However, it is possible to give more information about a vocabulary property using the `ESProperty` class, such as in the following example:

```
public ESProperty(String attrName, String attrType, ESValue
    defValue, boolean isMandatory, boolean isMultiValued, int
    rangeKind, int minRange, int maxRange, int index)
```

where the above terms have the following definitions:

- `attrName` —the name of the property
- `attrType` —the type of the property. The type could be any one of the fourteen data types supported.
- `defValue` —the default value assigned to the property.

- **isMandatory** —specifies whether the property is an essential property or an optional property. If the property is an essential property, any services registered in the vocabulary has to specify a value for this property during registration. A 'true' for this parameter means that the attribute is mandatory, and a 'false' means that it is not. This works only for an in-memory case. When the system is operated in JDBC mode, all the attributes are considered non-essential.
- **isMultiValued** —specifies whether the property can have multiple values or not. A 'true' for this parameter states that the property is multivalued and a 'false' informs the core that the property is single-valued.
- **rangeKind, minRange and maxRange** —specify what values that the property can take. The **rangeKind** can be one of the following: `ESConstants.NO_RANGE`, `ESConstants.LEFT_RANGE`, `ESConstants.RIGHT_RANGE`, and `ESConstants.FULL_RANGE`. The possible values that the attribute can take is determined by **minRange**, **maxRange**, and **rangeKind** parameters. The **rangeKind** parameter determines whether the range is to the right of **maxRange**, to the left of **minRange**, or occupies the full range.
- **index** —specifies whether the property should be used for indexing the database and if so what type of indexing. This can take the following values: `ESConstants.NO_INDEX`, `ESConstants.HASH_INDEX` and `ESConstants.TREE_INDEX`. Hash or Tree indexing should be chosen based on the kind of queries that is likely to be made. If the queries are likely to be equality queries (`"PrinterName == 'Acme'"`), it is suggested that hash indexing is used. If the queries are likely to be range constraints (`"Speed < 10"`), then tree indexing gives better performance. Hash or tree indexing makes a difference only in in-memory mode. In JDBC mode, they are ignored and standard indexing provided by the databases is used.

To create a new vocabulary property and add it to the vocabulary description, use code similar to this example:

```
ESProperty newProp = new ESProperty("Location", "String", new
ESValue("Bldg5"), true, true, ESConstants.NO_RANGE, 0, 0,
ESConstants.NO_INDEX);

printVocabDescription.addProperty(newProp);
```

ESServiceDescription

The preceding example describes how a printer Vocabulary is defined. This Vocabulary description can be used by a standards body to register a Printer Vocabulary. Now, a printer manufacturer can choose to use this Vocabulary to advertise a printer with appropriate attribute values.

For example, if a Printer Service provider were interested in offering an "Acme" printer as a Service, it can choose to describe the printer as follows:

```
//describe printer in printervocab
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection =
    new ESConnection(propertyFileName);
ESVocabularyFinder printVocabFinder =
    new ESVocabularyFinder(coreConnection);
ESQuery printVocabQuery =
    new ESQuery("Name == 'PrintVocab'");
ESVocabulary printerVocab =
    printVocabFinder.find(printVocabQuery);

ESServiceDescription printDescription =
    new ESServiceDescription(printerVocab);
printDescription.addAttribute("Manufacturer", "HP");
printDescription.addAttribute("Modelname", "LP 5");
printDescription.addAttribute("DPI", (int) 1400);
```

Multi-Valued Attributes

One of the new features in version 3.0 is support for multi-valued attributes in in-memory systems. This support allows the user to specify multiple values for a single service attribute. The following example deals with a printer that will accept several types of documents, including .pdf, .ppt, and .doc. The following commands set the printer's DocumentType attribute to accept multiple values:

```
ESAttribute multiAttr = new ESAttribute("DocumentType", new
    ESValue("DOC"));
// add additional values
Object[] vals = new Object[3];
vals[0] = "PDF";
vals[1] = "PPT";
```

```
vals[2] = "XLS";
multiAttr.addValues(vals);
description.addAttribute(multiAttr);
```

If the client uses any of these values to do a lookup, the service will be found by the client. For example:

```
finder.find(vocab, new ESQuery("DocumentType == 'DOC'"));
finder.find(vocab, new ESQuery("DocumentType == 'XLS'"));
finder.find(vocab, new ESQuery("DocumentType == 'PPT'"));
```

All these queries will result in the service being found. This powerful feature can be used in cases where a service attribute may have more than one value as in the previous example.

XML Descriptions of Services

So far, APIs have been used to create and manage descriptions in Java. In addition to these APIs, J-ESI supports XML descriptions of Vocabularies and Service

s using the constructors described here. XML is the preferred mode for describing Vocabularies and Services. Typically, XML is used for data descriptions passed as parameters to the description constructors.

XML Vocabulary Description

The `ESVocabularyDescription` class has a constructor that takes in an `ESXMLFile` describing the Vocabulary encapsulated in an XML page:

```
public ESVocabulary(ESXMLFile xml_stream, ESConnection conn);
```

This example creates a new Vocabulary with three attribute properties: Manufacturer, Modelname, and DPI:

```
<?xml version="1.0"?>
<ESpeak version="E-speak 2.0" operation="CreateVocab"
xmlns="http://localhost/e:/Esxml/Schemas/espeak.xsd">
  <resource xmlns="">
    <resourceDes xmlns="">
      <pattern>
        <Name xmlns="">
          printervocab
        </Name>
```

```

        <Type>
          Vocabulary
        </Type>
      </pattern>
    </resourceDes>
    <resourceData xmlns="">
      <attrGroup name="my printer Vocabulary" xmlns="">
        <attrDecl xmlns="" name="Manufacturer" required="no">
          <datatypeRef xmlns="" name="string"/>
        </attrDecl>
        <attrDecl xmlns="" name="Modelname" required="no">
          <datatypeRef xmlns="" name="string"/>
        </attrDecl>
        <attrDecl xmlns="" name="DPI" required="no">
          <datatypeRef xmlns="" name="integer">
        </datatypeRef>
        </attrDecl>
      </attrGroup>
    </resourceData>
  </resource>
</ESpeak>

```

All XML requests in e-speak have E-speak as the root element, which specifies the version information as well as the intended operation for the Services described in the request. As noted earlier, two aspects are required to describe Vocabularies.

First, how Clients can find the Vocabulary has to be described. It is assumed that the Vocabularies are themselves defined in the default Vocabulary that allows specifying, among other things, the name and type of Service. In the preceding XML description, the element `resourceDes` specifies the Name of this Vocabulary to be `printervocab`. Because this is a Vocabulary, its Type is Vocabulary.

Second, the properties of the Vocabulary must be specified. The properties that make up the printer Vocabulary are contained in an `attrGroup` element. The `attrGroup` element contains a list of `attrDecl` elements, each describing the specifics of an attribute property, including its name and data type.

The XML file defines a Printer Vocabulary that has three properties:

- 1 Manufacturer
- 2 Modelname
- 3 DPI.

As with queries, J-ESI 3.0 supports a new XML format for creating vocabularies. The new format involves constructing `ESVocabularyDescription` with two `ESXMLFile` arguments

```
public ESVocabularyDescription(ESConnection connection,
                               ESXMLFile xmlHeader, ESXMLFile xmlRequest)
```

The contents of `xmlHeader` looks as follows:

```
<?xml version='1.0'?>
<header xmlns="http://www.e-speak.net/Schema/E-
speak.header.xsd">
  <communication>
    <to>es://localhost:12346/WebAccess/CreateVocab</to>
  </communication>
</header>
```

The contents of `xmlRequest` looks as follows:

```
<?xml version='1.0'?>
<resource xmlns="http://www.e-speak.net/Schema/E-speak.register.xsd">
  <resourceDes>
    <vocabulary>http://www.e-speak.net/Schema/E-speak.base.xsd</
vocabulary>
    <attr name="Name">
      <value>printervocab</value>
    </attr>
    <attr name="Type">
      <value>Vocabulary</value>
    </attr>
  </resourceDes>
  <attrGroup name="printervocab" xmlns="http://www.e-speak.net/Schema/
E-speak.vocab.xsd">
    <attrDecl name="Model" required="true">
      <datatypeRef name="String"/>
    </attrDecl>
    <attrDecl name="DPI" required="true">
      <datatypeRef name="String"/>
    </attrDecl>
    <attrDecl name="Manufacturer" required="true">
      <datatypeRef name="String"/>
    </attrDecl>
  </attrGroup>
</resource>
```

XML Service Description

The `ESServiceDescription` class has a constructor that takes in an ESXML file describing the Vocabulary encapsulated in an XML page, as follows:

```
public ESServiceDescription(ESXMLFile xmlDescriptionFile,
    ESConnection coreConnection);
```

This is the recommended mode of creating and registering Services for compatibility with existing or emerging Vocabularies. The following example registers a new Service using the Vocabulary just created:

```
<?xml version="1.0"?>
<ESpeak version="E-Speak 1.0beta" operation="RegisterService"
xmlns="http://localhost/e:/Esxml/Schemas/espeak.xsd">
  <resource>
    <resourceDes xmlns="" name="Printer">
      <!-- Specify printer Vocabulary -->
      <query xmlns="">
        <queryBlock xmlns="">
          <WHERE xmlns="">
            <!-- absence of query implies Base Vocabulary -->
            <condition xmlns="">
              <IN xmlns="">
                <pattern xmlns="">
                  <Name
                    xmlns="">printervocab</Name>
                  <Type
                    xmlns="">Vocabulary</Type>
                </pattern>
              </IN>
            </condition>
          </WHERE>
        </queryBlock>
      </query>
      <!-- Begin: attributes -->
      <attrSet xmlns="">
        <!-- End: Use printerVocabulary -->
        <attr xmlns="" name="Manufacturer" required="true">
          <value xmlns="">HP</value>
        </attr>
        <attr xmlns="" name="Modelname" required="false">
          <value xmlns="">LP 5</value>
        </attr>
```

```

        <attr xmlns="" name="DPI">
          <value xmlns="">1400</value>
        </attr>
      <!-- End: attributes -->
    </attrSet>
  </resourceDes>
</resource>
</ESpeak>

```

In the preceding example, XML has been used to describe a specific printer. The XML description has two parts. The first part of the XML document, the Vocabulary in which the rest of the description is to be interpreted, is specified. For example, the query specifies that the following description is in a Vocabulary whose name is `printervocab`:

```

<?xml version="1.0"?>
<ESpeak version="E-Speak 1.0beta" operation="RegisterService"
xmlns="http://localhost/e:/Esxml/Schemas/espeak.xsd">
  <resource>
    <resourceDes xmlns="" name="Printer">
      <!-- Specify printer Vocabulary -->
      <query xmlns="">
        <queryBlock xmlns="">
          <WHERE xmlns="">
            <condition xmlns="">
              <IN xmlns="">
                <pattern xmlns="">
                  <Name>printervocab</Name>
                  <Type>Vocabulary</Type>
                </pattern>
              </IN>
            </condition>
          </WHERE>
        </queryBlock>
      </query>
    </resourceDes>
  </resource>
</ESpeak>

```

In the second part, the element `attrSet` actually contains the set of attribute value pairs that describe the particular Service in question. The following example describes a printer whose Manufacturer is HP, whose Model name is LP 5, and whose DPI is 1400:

```

<?xml version='1.0'?>
<resource xmlns="http://www.e-speak.net/Schema/E-
speak.register.xsd">

```

```

<resourceSpec>
  <locator>
    http://www.hp.com
  </locator>
</resourceSpec>
<resourceDes>
  <vocabulary>
    printervocab
  </vocabulary>
  <attr name="Manufacturer">
    <value>HP</value>
  </attr>
  <attr name="Modelname">
    <value>LP 5</value>
  </attr>
  <attr name="DPI">
    <value>1400</value>
  </attr>
</resourceDes>
</resource>

```

J-ESI does support the older XML schemas for registering services, but they should be treated like deprecated schemas and clients are encouraged to move to the newer schemas. The header for the service description creation looks as follows:

```

<?xml version='1.0'?>
<header xmlns="http://www.e-speak.net/Schema/E-
speak.header.xsd">
  <communication>
    <to>es://localhost:12346/WebAccess/RegisterService</to>
    <context>
      <!-- session token is inserted by the requesting appl. -->
    </context>
  </communication>
</header>

```

The ESServiceDescription constructor that uses the new XML entry points looks as follows:

```

public ESServiceDescription(ESConnection connection,
    ESXMLFile xmlHeader,
    ESXMLFile xmlRequest)

```

75

```

ESConnection coreConnection = new
ESConnection(propertyFileName);
ESXMLFile xmlDescriptionFile =
    new ESXMLFile("/users/printer.xml");
ESServiceDescription printDescription =
    new ESServiceDescription(xmlDescriptionFile,
        coreConnection);
ESServiceElement printElement =
    new ESServiceElement(coreConnection, printDescription);
printElement.setImplementation(new PrinterServiceImpl());
ESAccessor printAccessor = printElement.register();
printElement.start();

```

In this example, a single thread is started that handles the requests to the Service. All requests destined to this Service are dispatched to an instance of the `PrinterServiceImpl` object.

There are situations where Clients may want to associate multiple Services with the same thread, or have multiple threads for the same Service. The current Client library allows Service providers to control the number of threads through the notion of a Service handler, represented by the `ESServiceHandler` class.

Every connection with the e-speak Core has a default Service handler associated with it. This default Service handler can be retrieved using the `getDefaultServiceHandler` call in the connection, as shown by the following code fragment:

```

String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceHandler defaultHandler =
    coreConnection.getDefaultServiceHandler();

```

In essence, each Service handler encapsulates a channel of communication through which messages are received. A single Service handler can serve as a channel for getting messages to multiple Services. Each Service handler also has control over the number of threads that process the messages that are delivered on the communication channel.

In the following example, a print server maintains a message queue for multiple printers. This is made possible by using the same handler for all the printers. The print server has 32 threads that process the requests of the Clients. The following code sample shows how this is accomplished:

008027 120800

```

String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceHandler printHandler =
    new ESServiceHandler(coreConnection);
printHandler.setNumThreads(32);

ESServiceDescription printDescription1 =
    new ESServiceDescription();
printDescription1.addAttribute("Name", "Printer1");
ESServiceElement printElement1 = new
    ESServiceElement(coreConnection, printDescription1);
printElement1.setImplementation(new PrinterServiceImpl());
printElement1.setHandler(printHandler);
ESAccessor printAccessor1 = printElement1.register();
printElement1.start();

ESServiceDescription printDescription2 =
    new ESServiceDescription();
printDescription2.addAttribute("Name", "Printer2");
ESServiceElement printElement2 =
    new ESServiceElement(coreConnection, printDescription2);
printElement2.setImplementation(new PrinterServiceImpl());
printElement2.setHandler(printHandler);
ESAccessor printAccessor2 = printElement2.register();
printElement2.start();

```

The Service elements for each printer share the same handler. This way, there is a single queue for requests for both printers, and there are 32 threads that handle requests to the printers.

Service-Specific Data

As described earlier, Service-specific data is arbitrary data that can be associated with Services. Service-specific data comes in two forms: public and private.

PublicData is data that can be accessed by any Client that can find the Service.

PrivateData is data that is sent back to the Service handler along with each request to the Service. For example, this data can be used by the Service handler to distinguish among various Services that share this handler.


```

try{
    ESConnection connection = new ESConnection("file.pr");

    //Creating the vocabulary.....
    ESVocabularyDescription vocabDesc = new
    ESVocabularyDescription();
    ESVocabularyDescription vocabDesc1 = new
    ESVocabularyDescription();

    //Setting the vocabulary name
    vocabDesc.addAttribute("Name","vocab");
    vocabDesc1.addAttribute("Name","vocab1");

    //Adding vocabulary properties
    vocabDesc.addStringProperty("printerSpeed");
    vocabDesc1.addStringProperty("printerDPI");

    //Registering the vocabulary
    ESVocabularyElement vocabElem = new
    ESVocabularyElement(connection,vocabDesc);
    ESVocabularyElement vocabElem1 = new
    ESVocabularyElement(connection,vocabDesc1);
    ESVocabulary vocab = vocabElem.register();
    ESVocabulary vocab1 = vocabElem1.register();
    ESServiceDescription sd = new ESServiceDescription[2];
    //Creating the service description
    sd[0] = new ESServiceDescription(vocab);
    sd[1] = new ESServiceDescription(vocab1);

    //Setting the service name
    sd[0].addAttribute("Name","myPrinter");
    sd[0].addAttribute("printerSpeed", "10");
    sd[1].addAttribute("Name","myPrinter");
    sd[1].addAttribute("printerDPI", "1000");
    //Create the service element
    ESServiceElement se = new ESServiceElement(connection,
    sd);

```

Restarting Existing Services

Just as Service providers can start new Services, J-ESI allows Service providers to restart existing Services. Clients may want to restart Services in order to resume a Service after recovering from a Service shutdown or outage. Clients can restart

Services in three ways: (i) They can construct a Service element from the description and call `restart()` on the element, (ii) they can store away the accessor for the service in a persistent folder and use the accessor to restart the service, or (iii) they can store away the accessor of the handler in a persistent folder and use that accessor to restart the service. Modes (ii) and (iii) are discussed in the next chapter in the section on folders.

The element attempts to find a Service whose description matches the description currently provided in the element. If such a Service is found, it activates that Service so that requests to that Service are now handled. However, if no such Service is found, an exception occurs and the Client is expected to catch the exception and then register and start the Service.

However, the recommended mode for restarting Services is with the use of Folders. The Client is expected to have created a local name for the Service in their folder hierarchy. See "Managing Bindings Using Folders" on page 91 to determine how to restart Services using the binding.

The following example shows the typical use of the restart method without the use of Folders:

```
public static void main (String [] args)
{
    try
    {
        String propertyFileName =
            new String("/users/connection.prop");
        ESConnection coreConnection =
            new ESConnection(propertyFileName);
        ESXMLFile xmlDescriptionFile =
            new ESXMLFile("/users/printer.xml");
        ESServiceDescription printDescription =
            new ESServiceDescription
                (xmlDescriptionFile, coreConnection);
        ESServiceElement printElement =
            new ESServiceElement(coreConnection, printDescription);
        printElement.setImplementation(new PrinterServiceImpl());
        ESServiceHandler printHandler =
            new ESServiceHandler(coreConnection);
        printHandler.setNumThreads(2);
        printElement.setHandler(printHandler);
        try
```

```

        {
            printElement.restart();
            System.out.println("XMLPrintServer re-started");
        }
        catch (ESLibRuntimeException ere)
        {
            printElement.register();
            printElement.start();
            System.out.println("XMLPrintServer started");
        }
    }
    catch (Exception e)
    {
        // handle the exception
    }
}

```

Registering Vocabularies and Contracts

Vocabularies and Contracts are themselves Services that are registered with the e-speak infrastructure. However, the Service provider for Vocabularies and Contracts is the e-speak Core itself. Because of this, Clients who create Vocabularies only register them, and do not start a thread in order to serve requests to the Vocabulary or Contract.

The classes that allow Clients to register Vocabularies and Contracts are `ESVocabularyElement` and `ESContractElement`, respectively. The following code fragment registers a printer Vocabulary that has three properties: DPI, Manufacturer, and Modelname:

```

public class VocabularyCreator
{
    public static void main(String[] argv)
    {
        try
        {
            String propertyFileName =
                new String("/users/connection.prop");
            ESConnection coreConnection =

```

```

        new ESConnection(propertyFileName);

    ESVocabularyDescription printVocabDescription =
        new ESVocabularyDescription();
    printVocabDescription.addAttribute
        ("Name", "printervocab");
    printVocabDescription.addIntegerProperty("DPI");
    printVocabDescription.addStringProperty("Manufacturer");
    printVocabDescription.addStringProperty("Modelname");

    ESVocabularyElement printVocabElement =
        new ESVocabularyElement(coreConnection,
            printVocabDescription);
    ESVocabulary printVocab = printVocabElement.register();

    Property[] propertyList = printVocab.getProperties();
    for (int i = 0; i < propertyList.length; i++)
    {
        System.out.println(propertyList[i].getPropertyName());
    }
    catch (Exception e)
    {
        // handle the exception
    }
}
}

```

Creating Contracts is very similar to the creation of Vocabularies. The following code fragment shows the creation of a simple Contract:

```

public class ContractCreator
{
    public static void main(String[] argv)
    {
        try
        {
            String propertyFileName =
                new String("/users/connection.prop");
            ESConnection coreConnection =
                new ESConnection(propertyFileName);

```

```

ESContractDescription printContractDescription =
    new ESContractDescription();
printContractDescription.addAttribute
    ("Name", "PrintContract");
    String printIDL = getIDLFromFile();
printContractDescription.setInterfaceDefinition(printIDL);

ESContractElement printContractElement =
    new ESContractElement(coreConnection,
        printContractDescription);
ESContract printContract = printContractElement.register();

String receivedPrintIDL =
    printContract.getInterfaceDefinition();

System.out.println(receivedPrintIDL);
}
catch (Exception e)
{
    // handle the exception
}
}

```

On registering a Vocabulary or Contract, the Clients receive a stub to the created Vocabulary or Contract. This is in contrast to other Services that the Client creates, where they receive an accessor to the Service that they create. (ESAccessor is explained in detail in the later sections) This is because the Client that is registering the Vocabulary or Contract is not the handler for that Vocabulary or Contract.

A Bank Service Example

The following example is a complete Bank Service example that makes use of many of the concepts introduced in this chapter. Figure 11 shows the relationship between a Bank Service, a Bank Service Contract, and the Bank Service Vocabulary.

Typically, the Bank Vocabulary, Bank Contract, and Bank Service are not defined by the same programmer; however, for the sake of simplicity in this example, we can assume that they are defined by the same person.

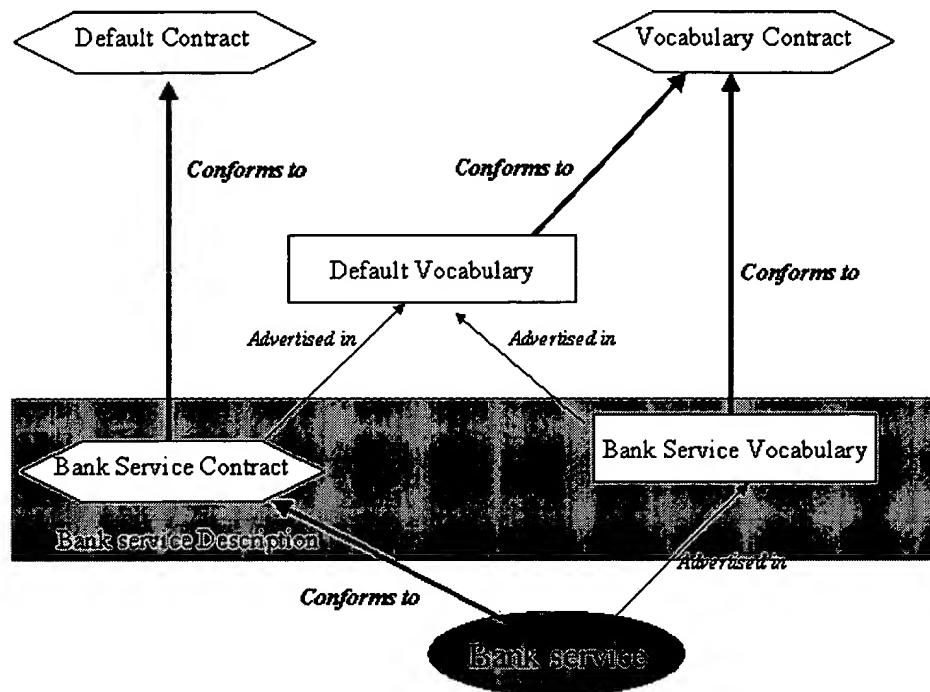


Figure 11 Example of a Bank Service

The following example lists the sequence of actions performed by a Bank Service (assuming the Vocabulary, Contract, and Service are all defined by the same piece of code):

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESContractDescription bankContractDescription =
    new ESContractDescription();
bankContractDescription.
    addAttribute("Name", "BankContract");
```

```

ESContractElement bankContractElement =
    new ESContractElement(coreConnection,
        bankContractDescription);
ESContract bankContract = bankContractElement.register();

ESXMLFile xmlDescriptionFile =
    new ESXMLFile("/users/bankVocab.xml");
ESVocabularyDescription bankVocabDescription = new
    ESVocabularyDescription(xmlDescriptionFile,
        coreConnection);
ESVocabularyElement bankVocabElement = new
    ESVocabularyElement(coreConnection,
        bankVocabDescription);
ESVocabulary bankVocab = bankVocabElement.register();

xmlDescriptionFile =
    new ESXMLFile("/users/bank.xml");
ESServiceDescription bankDescription =
    new ESServiceDescription(xmlDescriptionFile,
        coreConnection);
bankDescription.setContract(bankContract);

ESServiceElement bankElement =
    new ESServiceElement(coreConnection, bankDescription);
bankElement.setImplementation(new BankServiceImpl());
ESAccessor bankAccessor = bankElement.register();
bankElement.start();

```

The following example shows the contents of the bankVocab.xml file that is used to describe the Vocabulary:

```

<?xml version="1.0"?>
<ESpeak version="E-speak 1.0" operation="CreateVocab">
  <resource>
    <!-- Begin: Specify the Vocabulary description -->
    <resourceDes>
      <attrSet>
        <attr name="Name">
          <value>bankvocab</value>
        </attr>
      </attrSet>
    </resourceDes>
    <!-- End: Specify the Vocabulary description -->
  </resource>
</ESpeak>

```

```

<resourceData>
  <!-- Begin: Specify the attribute property set -->
  <attrGroup name="Bank Vocabulary">
    <attrDecl name="Name">
      <datatypeRef name="string"/>
    </attrDecl>
    <attrDecl name="createAccount">
      <datatypeRef name="string"/>
    </attrDecl>
    <attrDecl name="tradeStock">
      <datatypeRef name="string"/>
    </attrDecl>
  </attrGroup>
  <!-- End: Specify the attribute property set -->
</resourceData>
</resource>
</Espeak>

```

The following example lists the contents of the bank.xml file that is used to describe the particular Bank Service:

```

<?xml version="1.0" ?>
<ESpeak version="E-Speak 1.0" operation="RegisterService">
  <resource>
    <resourceDes name="Bank Description">
      <!-- Begin: Specify bank Vocabulary in a query-->
      <query>
        <queryBlock>
          <WHERE>
            <condition>
              <IN>
                <pattern>
                  <Name>bankvocab</Name>
                </pattern>
              </IN>
            </condition>
          </WHERE>
        </queryBlock>
      </query>
      <!-- End: Specify bankVocabulary -->
      <!-- Begin: the attribute list -->
      <attrSet>
        <!-- End: Use bank Vocabulary -->
        <attr name="Name">

```



```

        <value>Acme</value>
    </attr>
    <attr name="createAccount">
        <value type="string">"Acme 1"</value>
    </attr>
    <attr name="tradeStock">
        <value type="string">"hwp"</value>
    </attr>
</attrSet>
<!-- End: the attribute list -->
</resourceDes>
</resource>
</ESpeak>

```

Accessing Descriptions: ESAccessor

ESAccessor is a reference to a Service with which the user can do the following.

- Obtain or change the attributes of a Service
- Send messages to the Service (More details are in Appendix B)

Typically, a Client finds a Service that matches some desired attributes, but after finding the Service, the Client may be interested in checking the values of other attributes as well.

For example, a Client may find a list of printers meeting a certain DPI value, but after finding this list, the Client may want to determine the manufacturer name before proceeding further. In such situations, the Client can get the accessor for each Service and query the accessor for the manufacturer attribute's value.

Similarly, an administrator who manages the printer may want to mutate one of the searchable attributes or add new Data to reflect some upgrade to the printer. In this case, the ESAccessor is used to mutate the Service description as well. For example:

```

String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESXMLFile xmlDescriptionFile =

```

```

    new ESXMLFile("/users/printerQuery.xml");
    ESXMLQuery printQuery = new ESXMLQuery(coreConnection,
    xmlDescriptionFile);
    ESServiceFinder printFinder = new
    ESServiceFinder(coreConnection);
    ESService printService = printFinder.find(printQuery);
    ESAccessor printAccessor =
    ((ESAccessorHandle)printService).getAccessor();

```

On the Service provider side, when a Service is registered, an accessor is returned as a result of doing the register call. For example:

```

String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceDescription printDescription =
    new ESServiceDescription(printVocab);
printDescription.addAttribute("Modelname", "HP1000");
ESServiceElement printElement =
    new ESServiceElement(coreConnection, printDescription);
printElement.setImplementation(new PrinterServiceImpl());
ESAccessor printAccessor = printElement.register();
printElement.start();

```

The Service provider can now use the accessor to access and mutate the description of the Service.

Consider a printer Client who, after finding a printer with DPI == 1400, decides to list all the attributes of the printer, perhaps in an effort to find out its manufacturer, speed, and so on.:

```

String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceFinder printFinder =
new ESServiceFinder(coreConnection);
ESQuery printQuery = new ESQuery(printervocab);
printQuery.addConstraint("DPI==1400");
ESService printService = printFinder.find(printQuery);
ESAccessor printAccessor =
    ((ESAccessorHandle)printService).getAcessor();
System.out.println("Attributes of this printer are:\n");
ESAttribute[] attrList =printAccessor.getAttributes();
for(int i=0; i<attrList.length; i++)

```

```

    {
        System.out.println(attrList[i].toString());
    }

```

A Service provider who has created a Service or a Client and who has the appropriate permissions can change the attributes of the Services. For example, if the administrator upgrades the printer to the latest model that supports a higher DPI, the administrator can update the description of the printer with the new values for the attributes as follows:

```

// code to find print service from above
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESVocabularyFinder printVocabFinder =
    new ESVocabularyFinder(coreConnection);
ESQuery printVocabQuery =
    new ESQuery("Name == 'printervocab'");
ESVocabulary printVocab =
    printVocabFinder.find(printVocabQuery);

ESQuery printQuery = new ESQuery("Manufacturer == 'HP'");
ESService printService = printFinder.find(printQuery);
ESAccessor printAccessor =
    ((ESAccessorHandle)printService).getAccessor();

ESAttribute attr1 = new ESAttribute("Manufacturer");
attr1.setValue("HP");
ESAttribute attr2 = new ESAttribute("DPI");
attr1.setValue((int) 1400);

printAccessor.setAttribute(attr1, printVocab);
printAccessor.setAttribute(attr2, printVocab);

```

The ESAccessor class also has other methods that can be used to mutate the metadata of services. Examples of such methods are the following. Note that though some of these signatures refer to ESBaseDescription, in practice, these return values or arguments to these functions are instances of ESVocabularyDescription, ESContractDescription, ESViewDescription, ESServiceDescription, etc.

```

public void setDescriptions(ESBaseDescriptions [] desc);

```

0930-7688

Chapter 4 Extended Services

The previous chapters described the programming model, and how e-speak Clients and Services interact. In addition to support for these basic functions, J-ESI supports extended Services that enable support for persistent bindings, Event-based interaction semantics, and loosely coupled distributed Communities. This chapter shows how these extended Services can be used to create sophisticated Service interactions.

The chapter is divided into the following sections:

- Managing Bindings Using Folders
- Repository Views
- Categories
- Communities
- Security
- Events

Managing Bindings Using Folders

One of the main reasons why e-speak can support loosely coupled distributed Services is that it does not rely on global naming. In many traditional distributed systems, the distribution is possible because Clients know the name of the Service providers and they use the name to access the Service.

In e-speak, on the other hand, there are no global names. Clients can make up names for the Services they find or create, and that name is independent of the name that another Client uses for the same Service.

For instance, one Client may refer to a Printer Service as "Marketing Printer," while another Client may refer to the same Printer Service as "Engineering Printer." This allows Services to be migrated or upgraded, without having to change the Client-side programs that use these Services. Figure 12 illustrates the naming process in e-speak.

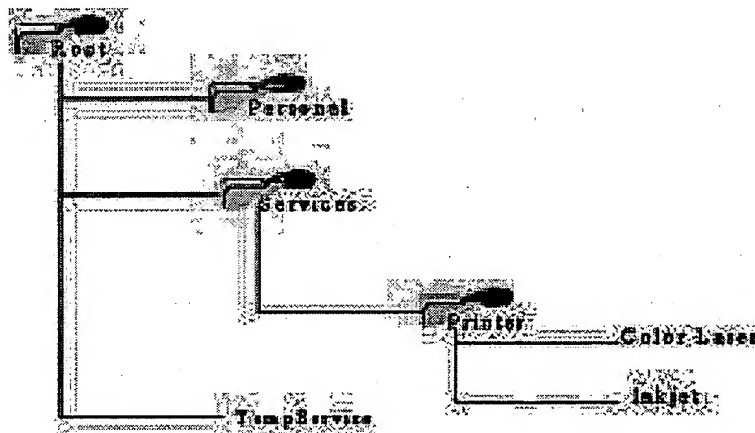


Figure 12 Example of folder names using e-speak

In J-ESI, Clients manage their local name spaces using folders. Folders are analogous to directories in traditional file systems. Clients can create bindings between the names and Services they find and then put the bindings in a folder.

Essentially, folders enable Clients to build a local hierarchical name space. Every user account in an e-speak Core has a folder that is its root (typically denoted by /). Clients can get at the root folder in their session by invoking the `getRootFolder` method in `ESConnection`. For example:

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
    ESConnection(propertyFileName);
ESFolder myRoot = coreConnection.getRootFolder();
```

Creating Folders

Folders created by Clients are either persistent or transient. A transient folder is a folder that does not survive beyond the lifetime of the connection in which it is created. A persistent folder, on the other hand, can survive beyond the lifetime of the connection in which it is created. If the Core is backed up in a database, persistent folders also survive Core reboots. This is because, although the folders are a Client-side abstraction, their state is maintained in the e-speak Core.

Creating Transient Folders

Currently, Clients can create transient or persistent folder hierarchies under the root folder. To create a transient folder hierarchy, they create a subfolder of the root folder that is transient. For example, to create a transient folder under the root folder called bookmarks, use the following constructor of ESFolder:

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESFolder bookMarkFolder = new ESFolder(coreConnection,
"bookmarks");
```

This call creates a transient folder under the root folder called /bookmarks. The folder constructor is used to create subfolders of the root folder, while the createSubFolder method is used to create subfolders of all non-root folders.

Creating Persistent Folders

The above ESFolder constructor cannot be used to create persistent folders. To create a persistent folder called /myServices under the root folder, the following call is used:

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESFolder ownFolder = new ESFolder(coreConnection, "myServices",
true);
```

As in most operating systems, there is the idea of a current folder in J-ESI. Clients can get their current folder through the Service context, as shown by the following code fragment:

When a new connection is created, the name of the home folder is read from the properties file that is passed to the constructor of `ESConnection`. The value of the property named `homeFolder` in the properties file is used as the name of the home folder. If no properties file is passed to the constructor of `ESConnection`, the default home folder is `/home`. The home folder can be obtained as follows:

The properties of the current folder that the Client is located in are important. If a Client creates Services while in a persistent folder, the Service is created as a persistent Service. This means that the description of the Service is registered with the e-speak Core even if the handler of the Service disconnects from the e-speak Core.

Clients typically create folders to manage name bindings on Services they have discovered or created. For example, an administrator may add bindings to discovered print Services in a persistent folder (such as `/home/services/printers`). Because the bindings are stored in a persistent folder, anytime the administrator reconnects, they continue to have access to previously discovered printers by simply looking into the persistent `/home/printers` folder.

The following code fragment shows the creation of a subfolder 'printers' of the Client's home folder (/home/services). After printers is created, the Client finds a printer and places a name binding for it in this folder.

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceContext connectContext =
    coreConnection.getServiceContext();
ESFolder currentFolder = connectContext.getCurrentFolder();
// The current folder is assumed to be /home/services.
// This can be set in the properties file
ESFolder newFolder = null;
try {
    newFolder = currentFolder.getSubFolder("printers");
}
catch (InvalidNameException ine){
    newFolder = currentFolder.createSubFolder("printers");
}
connectContext.setCurrentFolder(newFolder);
PrinterServiceIntf printer = null;
try{
    printer = (PrinterServiceIntf)
        newFolder.getService("myprinter", "PrinterServiceIntf");
}
catch (ESInvocationException esie){
    ESXMLFile xmlQueryFile = new ESXMLFile("/users/
printerQuery.xml");
    ESXMLQuery printQuery =
        new ESXMLQuery(coreConnection, printQuery);
    String intfName = PrinterServiceIntf.class.getName();
    ESServiceFinder printFinder =
        new ESServiceFinder(coreConnection, intfName);
    printer = (PrinterServiceIntf) printFinder.find(printQuery);
    newFolder.add("myprinter", printer);
}
// get the contents of the folder
String[] contents = newFolder.listNames();
Naming Created Services
Naming a Service which is created and adding it to a folder is
similar to naming a found Service. The following code shows an
example of how to name a created Service.
String propertyFileName = new String("/users/connection.prop");
```

09733027 120800

```

ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceContext connectContext =
    coreConnection.getServiceContext();
ESFolder currentFolder = connectContext.getCurrentFolder();
// The current folder is assumed to be /home/services.
// This can be set in the properties file
ESFolder newFolder = null;
try {
    newFolder = currentFolder.getSubFolder("printers");
}
catch (InvalidNameException ine){
    newFolder = currentFolder.createSubFolder("printers");
}
connectContext.setCurrentFolder(newFolder);
if (!(newFolder.containsName("fastprinter"))){
    ESXMLFile xmlDescriptionFile =
        new ESXMLFile("/users/printer.xml");
    ESServiceDescription printDescription =
        new ESServiceDescription(xmlDescriptionFile,
coreConnection);
    ESServiceElement printElement = new ESServiceElement(
        coreConnection, printDescription);
    printElement.setImplementation(new PrinterServiceImpl());
    ESServiceHandler essh = new ESServiceHandler(coreConnection);
    printElement.setHandler(essh);
    ESFolder homeFolder = coreConnection.getHomeFolder();
    homeFolder.add(HANDLER_NAME, essh.getAccessor());
    ESAccessor printAccessor = printElement.register();
    printElement.start();
    newFolder.add("fastprinter", printAccessor);
}
else
{
    ESServiceElement printElement = new
        ESServiceElement(newFolder.getAccessor("fastprinter"));
    printElement.setImplementation(new PrinterServiceImpl());
    printElement.restart();
}

```

When the Service provider goes off-line and logs back into the e-speak Core, they do not have to re-create the Service. The provider looks for the name binding corresponding to the printer that they created and performs a restart. This allows the print Service to go online and to respond to requests.

```
ESConnection conn = new ESConnection("propfile");
ESFolder home = conn.getHomeFolder();
    ESAccessor da = home.getAccessor(HANDLER_NAME);
    ESServiceHandler handler = new ESServiceHandler(da);
    handler.setNumThreads(numThreads);
    ESServiceElement element = new
ESServiceElement(handler);
    element.setImplementation(new PrintServiceImpl());
    element.restart();
```

Some of the constructors of `ESFolder`, as well as some of the `createSubFolder` calls in `ESFolder`, take an additional `ESServiceDescription` that describes the attributes of the folder that is being created. For example:

97

09-06-2017

```
ESXMLFile xmlDescriptionFile = new ESXMLFile("/users/  
folder.xml");  
ESServiceDescription folderDescription =  
    new ESServiceDescription(xmlDescriptionFile, coreConnection);  
ESFolder newFolder = new ESFolder("services",  
    folderDescription);
```

The preceding constructor of `ESFolder` causes the description of the folder to be registered in the Repository of the Core so that other Clients can discover the folder description. Folders are found using the `ESFolderFinder` class. For example:

```
String propertyFileName = new String("/users/connection.prop");  
ESConnection coreConnection = new  
    ESConnection(propertyFileName);  
ESFolderFinder fFinder = new ESFolderFinder(coreConnection);  
ESXMLFile xmlQueryFile =  
    new ESXMLFile("/users/folderQuery.xml");  
ESXMLQuery folderQuery =  
    new ESXMLQuery(coreConnection, xmlQueryFile);  
ESFolder folder = fFinder.find(folderQuery);
```

Navigating Folders

The `ESFolder` class has methods for navigating a folder hierarchy. For example, to get the subfolder of a given folder, there is a `getSubFolder` call. On the other hand, to get the parent of any folder, a `getParent` call is issued. In addition, there are other calls that allow the Client to list the contents of the folders.

Scopes

Scopes are used to mark the lifetime of transient and persistent Services.

Before scopes can be used, boundaries need to be defined for them. The following method in `ESServiceContext` is used to signify scope boundaries:

```
public void beginTransientScope();  
public void endTransientScope();
```

Ending a scope results in all the transient/persistent Services created in the scope being deleted. Furthermore, all the transient bindings for Services found by the Client are also deleted. For example:

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new
ESConnection(propertyFileName);
ESServiceContext connectContext =
coreConnection.getServiceContext();

...
while(notDone())
{
    connectContext.beginTransaction();
    ...
    createFindAndUseServices(coreConnection);
    ...
    connectContext.endTransientScope();
}
```

In this example, the `createFindAndUseServices` is a method that creates and uses a transient Service. The end scope call following this method invocation automatically deletes all transient bindings and Services created in the method.

Repository Views

The repository view is a feature in e-speak that allows clients to restrict the scope of their searches to within the view. These views are collections of e-speak registered e-speak services. Clients can add and remove services from these views. These views are maintained by the e-speak core engine, therefore they are core-managed services. These repository views are described using `ESViewDescription` just as vocabularies are described with `ESVocabularyDescription` and other services are described with `ESServiceDescription`. Similarly, clients use the `ESViewFinder` to find these repository views and create `ESViewElement` instances in order to register new repository views with the core repository.

The following code snippet shows how a client can create a repository view, find it, and add elements to it, etc.

```

public class testView
{
    public static void main(String args[])
    {
        ESConnection es = null;
        try {
            String propertyFileName = new String("/users/
connection.prop");
            es = new ESConnection(propertyFileName);
            ESViewDescription esd = new ESViewDescription();
            esd.addAttribute("Name", "TestView");
            ESViewElement ese = new ESViewElement(es, esd);
            ESView v = ese.register();
            ESViewFinder esf = new ESViewFinder(es);
            ESView[] ess = esf.findAll(new ESQuery("Name ==
'TestView'"));
            ESAccessor esa = ((ESAccessorHandle) ess[0]).getAccessor();
            ((ESViewStub)ess[0]).add(esa);
            boolean result = false;
            result = ((ESViewStub)ess[0]).contains(esa);
            // get all the accessors in the view
            ESAccessor[] e = ((ESViewStub)ess[0]).list();
            es.close();
        } catch (Exception e) {
        }
    }
}

```

Restricting searches with views

The typical use of such repository views is to restrict the search results from queries. This is accomplished by associating a view with a query. When an ESQuery with an associated view is executed by the e-speak repository, the search results are guaranteed to be from the elements within the repository view.

```

String propertyFileName = new String("/users/connection.prop");
ESConnection es = new ESConnection(propertyFileName);
    System.out.println("Connected to core");
ESViewFinder esf = new ESViewFinder(es);

```

```

        ESView view = esf.find(new ESQuery("Name == 'myPrintView'"));
        ESServiceFinder finder = new ESServiceFinder(es,
        "PrinterServiceIntf");
        ESVocabulary pVocab = // .. code to find vocabulary
        ESQuery query = new ESQuery(pVocab, "printerSpeed == '10'");
        query.setView(view);
        try {
            ESService[] list = finder.findAll(query);
            for (int i = 0; i < list.length; i++) {
                ESServiceStub printer = (ESServiceStub)list[i];
                ESAccessor desc = file.getAccessor();
            }
        }
        catch (LookupFailedException lfe) {
            // No entries were found that matched the search criteria
        }

```

Categories

J-ESI supports the notion of categories. These categories provide a way to classify services so that service providers can advertise in multiple categories and clients can find services of interest to them in the categories of interest to them. Each category is qualified by its name. Furthermore, one can create sub-categories of existing categories, and provide descriptions for them. When clients search for services, they too can set the categories that they want the search to be performed in. This becomes part of their query and picks out the services that are advertised in the categories of interest to the client.

The category list of interest to the client/service provider is maintained in the ESServiceContext that is associated with the ESConnection. The following example shows the registration of the print service in the "high speed printer"

category. Essentially, the service provider sets the current category list in the ESServiceContext and these categories are used when registering and advertising the service.

```
public static void main(String[] args)
{
    try {
        String propertyFileName = new String("/users/
connection.prop");
        ESConnection connection = new
        ESConnection(propertyFileName);
        ESCategoryFinder cf = new ESCategoryFinder(connection);
        ESCategory root = cf.findRootCategory();
        ESCategory cat1 = root.createCategory("high speed printer",
"high ppm");
        ESCategory[] catList = new ESCategory[2];
        catList[0] = root;
        catList[1] = cat1;
        connection.getServiceContext().setCategory(catList);
        ESServiceDescription sd = new ESServiceDescription();
        sd.addAttribute(ESConstants.SERVICE_NAME, "printer");
        sd.addAttribute("Description", "my hp printer");
        ESServiceElement se = new ESServiceElement(connection, sd);
        se.setImplementation(new PrinterServiceImpl());
        se.register();
        se.advertise();
        se.start();
    } catch (Exception e1) {
    }
}
```

Now, on the client side, the finder for the printers also uses the category list that is set in the service context. The code snippet below shows how to search for printers in the "high speed printer" category. Note that the client has to construct a category list that represents the path from the root category to the category of interest.

```
public static void main(String [] args) {
    try {
        String propFileName = args[1];
        ESConnection connection = new ESConnection( propFileName );
        ESCategoryFinder cf = new ESCategoryFinder(connection);
        ESCategory root = cf.findRootCategory();
```



```

ESAccessor[] cat1Acc = cf.find("high speed printer");
ESCategory cat1 = new ESCategoryStub(connection, cat1Acc[0]);
ESCategory[] catList = new ESCategory[2];
catList[0] = root;
catList[1] = cat1;
connection.getServiceContext().setCategory(catList);

ESQuery q = new ESQuery (ESConstants.SERVICE_NAME +
"=='printer'");
ESServiceFinder sf = new ESServiceFinder(connection,
                                           "PrinterServiceIntf");
ESService[] myObjs = sf.findAll(q);

if(( myObjs == null ) || (myObjs.length == 0))
    System.out.println( "ERROR IN CREATING STUB/FINDING" );
    System.out.println("Found " + myObjs.length + " services");

for(int i=0; i<myObjs.length; i++) {
    PrinterServiceIntf myObj =
        (PrinterServiceIntf)myObjs[i];
        String junk = myObj.print();
    }
    connection.close();
    return;
} catch (Exception e1) {
}
}

```

Delegators

Often, there is a need for the implementation objects associated with services to have access to some of the metadata of the services that the implementation object is associated with. For example, a file implementation is to be shared amongst multiple files that are registered as services with e-speak. Suppose furthermore, that the private service specific data is used to store the actual path to the contents of the file on disk. In particular, suppose that the name of the service specific data is "RealFileName". Essentially, when the implementation object extends the ESDelegatorImpl, the service specific data of the service is passed onto the implementation object. This allows the implementation object to determine the exact service for which this request is meant.

```

public class VFSFileImpl extends ESDelegatorImpl{
    final static String REAL_FILE_NAME = "RealFileName";

```

```
public VFSFileImpl(ESConnection connection, ESVocabulary vocab, ESLogClient
log,
                    int numThreads) {

// constructor....
}

public byte[] fetchBuffer(int offset, int size)
throws ESInvocationException {
    try {
        String fileName = getFileName();
        File file = new File(fileName);
        int count = size;
        int whatsLeft = (int)file.length() - offset;
        if (whatsLeft < count) {
            count = whatsLeft;
        }
        byte [] fileBuffer = new byte[count];
        FileInputStream in = new FileInputStream(file);
        in.skip((long)offset);
        in.read(fileBuffer);
        in.close();
        return fileBuffer;
    } catch (Exception ioe) {
    } finally {
    }
}

private String getFileName() {
    try {
        byte[] entry = getPrivateData(REAL_FILE_NAME);
        String fileName = new String(entry);
        return fileName;
    } finally {
    }
}
}
```

Communities

So far, the discussion has focused on how e-speak Clients can register and discover new Services while using folders to manage Service bindings. No mention has been made yet of multiple connected Cores, the distributed nature of deployments, or the impact of distribution on failure semantics, latency, and concurrency.

J-ESI makes it easy for programmers to write distributed Services. Although all Services are registered in the local Core by default, these can be made more widely visible by advertising this Service across multiple Cores. In J-ESI, the possible domains in which a Service is visible are as follows:

- **Only the e-speak Core**—In this deployment scenario, all Clients that want to use this Service are also connected to the same Core, loosely representative of a classic Client-server deployment of Services.
- **In an e-speak group**—An e-speak group is a collection of e-speak Cores that are closely connected to each other, such as in an administrative domain. These Cores typically can find all Services registered in any of the other Cores, and they may all share the same back-end server (possibly an LDAP server) for storing all Services registered in the group. E-services Village, a HP hosted service directory is an example of such a service directory. Such deployments are analogous to lookup or naming servers used in other solutions. Note: The advertising services without LDAP should use the same group name in the command line `-group <groupname>` option if they want to be part of the same group.
- **In an e-speak community**—An e-speak community is a Client-defined named set of e-speak groups that is created by Clients to enable them to search through different related sets of Services easily. The default community in J-ESI includes www.eservicesvillage.com. Therefore, every query that any client executes returns results that are not only in the local core, but also in eservices village.

In Figure 13, the top left corner shows a single Core case where all Clients and Service providers are connected to the same Core. The top right corner shows an e-speak group in which a closely connected group of Cores share an LDAP server or use other mechanisms to advertise all their Services to each other. The bottom

figure shows an e-speak community that can be formed between groups A and B. A Client in group A can select to find Services in group B, by setting the community list to include group B.

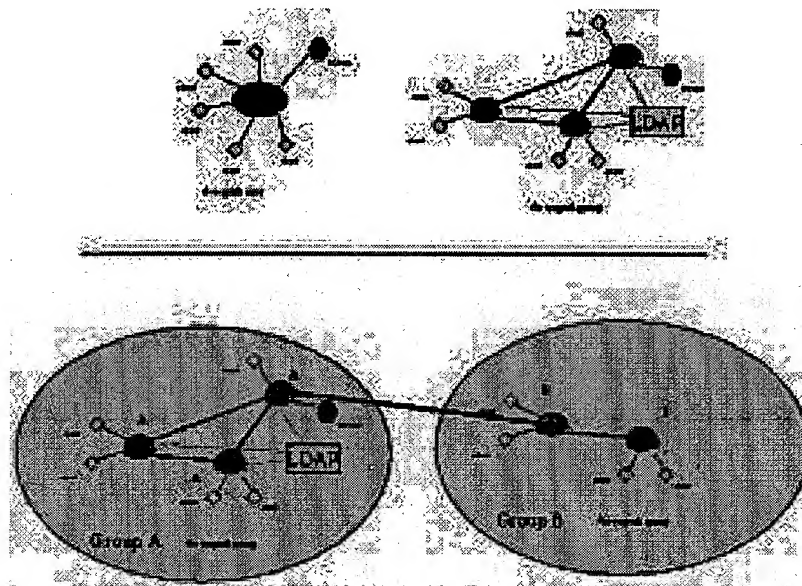


Figure 13 E-speak Core groups

A Service provider can advertise the Service in the local advertising service or can advertise it to different groups. The Service provider can do so by using either the `advertise` call or `advertiseInOtherGroups` method in `ESServiceElement`.

Consider an example where there are a number of printers organized according to administrative groups. Additionally, assume that each administrative group is associated with a different e-speak group. A user wanting to find printers across all administrative groups but still located closest to the user performs the following steps:

- 1 Create their own community that lists the groups of all the relevant administrative domains. Each group is identified by a string with the following format:

The host name and port number identify the host machine in which an advertising Service for the group of interest can be found. In addition, the name of the group of interest is specified as well. Note: The port is that of the connection factory and not of the Core. The advertising service should have been started with the same group name, with the command line option `-group <group-name>`.

Another way to create a community is to specify it in the properties file used on start-up. The properties file takes in a **community** property that is a comma-delimited list of groups such as:

```
...
community = host1:22020/auction_site1, host2:22021/auction_site2
...
```

- ```
ESServiceContext connectContext =
 coreConnection.getServiceContext();
connectContext.setCommunity(newCommunity);
```

- ```
ESServiceFinder anyFinder =
    new ESServiceFinder(coreConnection);
anyFinder.find("Name == 'AnyService'");
```

In fact, because e-speak is geared toward Service deployment, it is natural to simplify the process of advertising a Service across a wide geographical area, simplify searching, and provide distributed access control.

To simplify the programming of distributed Services, the following support is built into J-ESI:

- Advertising a Service automatically makes the Service visible to the group in which it has been advertised.
- Because each Core has a registry, an e-speak deployment does not require a centralized naming or lookup Service to obtain information about all Services registered. A centralized scheme can be implemented as a special case within a more general scheme enabled by e-speak finder Services. The search for a Service is automatically performed in parallel across multiple registries.

Advertising across internet and locally

Advertising service across the Internet using HP hosted global service directory

Service providers can advertise the services throughout the world using HP hosted global service directory by selecting default values when starting the advertising service. In this case, when user invokes the `advertise()` call, the service gets advertised in the global service directory. This allows a service provider located in Los Angeles to advertise a service to the global directory, accessible by any clients in New York. Clients and service providers can use the hosted gateway/connector service at eservicesvillage.com to access and provide services from behind a firewall.

For services hosted behind a firewall to advertise themselves at eservicesvillage.com (or any other service directory on the internet), the service provider has to set up the configuration files appropriately. The sections of the e-speak configuration file that pertain to web-proxy configuration and advertising service configuration may have to be modified in order for this to work correctly. The following is a sample section from an e-speak configuration file with the rules for web-proxy and advertising service configuration.

```
!-----
! Web proxy configuration
!-----
! webproxyname is a single value being the fully qualified hostname
! of the http-proxy used to traverse a firewall
```

```
net.espeak.infra.core.connector.webproxyname=web-proxy.efgijk.com
```

```
! webproxyport is the port on which the proxy listens
net.espeak.infra.core.connector.webproxyport=8088
```

```
! domain for which direct connection should be done.
! only one entry is supported currently
! Syntax is the end of the domain that should be match
! the '*' wildcard is not supported.
net.espeak.infra.core.connector.noproxydomain=efgijk.com
```

```
!-----
! Advertising Proxy configuration
!-----
!host at which the core for advertising proxy is running
net.espeak.services.advertise.esv_proxy_host=XXXXXXX
(needs to be fixed*****)
```

```
!port at which the core for advertising proxy is running
net.espeak.services.advertise.esv_proxy_port=23456
```

```
!default username and password to be used at the e-services village
net.espeak.services.advertise.esv_user_name=%esv_username%
net.espeak.services.advertise.esv_password=%esv_password%
```

If service providers who advertise their service have created an account at eservicesvillage.com, they can edit the `esv_user_name` and `esv_password` fields in the configuration file above to the appropriate values. This allows them to login to [eservicesvillage](http://eservicesvillage.com) and manage/edit their services.

Advertising a service within an enterprise

Service providers can advertise the services within an enterprise in two ways.

- Using a service directory (say LDAP)
- Using only the e-speak core repository

The first scenario is similar to using HP hosted global service directory except that the advertising service connects to the service directory specified by the service provider. This service directory can be located within the enterprise spread in different locations. Specifically, the advertising service is started by specifying (-

In the second scenario, service providers who do not want to use service directory like LDAP can still achieve the same results by starting the advertising service in 'With repository mode'. When the user invokes the `advertise()` call, the service is placed in the local advertising service.

Advertising a service in local domain

- Using a service directory (say LDAP)
- Using only the e-speak core repository

In the second scenario, the users can use the spontaneous discovery mechanism in the e-speak system. Service provider's advertisements are automatically transferred to all the advertising services belonging to the same group.

Multiple groups in a single service directory

Multiple groups can be in a single service directory (say LDAP). In this case, different advertising services belonging to different groups can connect to the same service directory and advertise services.

Selecting a group name

Two different service providers can advertise services with exactly the same descriptions (attribute values). If the service provider wants to prevent collisions across the advertised services or wants to protect the access to the services advertised, the service provider can specify sufficiently unique group name for the advertising service. Specifically the service provider uses (-group <groupname>) command line option of advertising service to achieve this. It is the service provider's and client's responsibility to select a sufficiently unique name for the groups to prevent collisions.

A client doing a search can specify a community which is a collection of group names. In this case, only services registered in the those groups are returned to the client. Services in other groups, even if matching client's query are not returned.

A service provider wishing to advertise the service to the whole world can do so by starting the advertising service with group name 'speaktome' and using the HP hosted global service directory. Specifically, the service provider starts the advertising service with command line option (-group "speaktome")

Setting Current Community

The following two Application Programming Interfaces (APIs) in the `ESServiceContext` class are used to create the community in which a search or a registration of a Service is to be performed.

The `getCurrentCommunity` and `setCommunity` methods are as follows:

```
public ESCommunity getCurrentCommunity();
public void setCommunity(ESCommunity community);
```

ESCommunity

To add an entry to the list of groups, use the following code:

```
public void add(String groupName);
```

The default community is defined by the start-up file.

To remove a group from the community, use the following code:

Example: `cs1demo5.rgv.hp.com:22022/group`.

where each element in `groupNames` is in the form `<hostname>:portnumber/name`.

```
ESServiceContext connectContext =
    coreConnection.getServiceContext();
connectContext.setCommunity(owncommunity);
.....
finder.find(query);
.....
```

Security

The most basic notion in security is the notion of identity that is determined by its key-pair. In order to invoke operations on a security-enabled service, a client requires an appropriate certificate. This certificate must be signed either by the service provider's principal himself or by another principal who is linked by a chain of delegation to a principal listed in the trust assumptions of the service provider. With security enabled in the core, service providers and service client require certificates to enable them access to core apis. For instance, to register a service and to perform find operations. The principal representing the core issues certificates to both client and provider enabling such access.

When security is enabled, start up of any e-speak client (service client or service provider) reads two files: the trust assumptions and the certificates. Both these are actually just lists of certificates: the former being used by a service provider and the latter being used by a service client. In the base case, the service provider requires only one certificate: that for accessing the core. The service client, on the other hand, requires one for accessing the core and one or more for accessing the service itself. The client certificate list is merely a concatenation of these. An example of a certificate that the core can provide a service provider looks as follows, the binary data has been truncated for brevity.

```
(signed (cert
  (issuer (public-key elgamal-pkcs1
    "\003\017v\245\235b\004\345\211\225\021[\203=\256/
    K\256\375\032\217:\351\024\327\304\342\312B\'\311\016\007\304^\2052\322\27
    1@\304`<\370\204\036j\220\030\2217aD*\242\335|\233H\334N\201?.Uq\236)"))
  (subject (public-key "elgamal-pkcs1"
    "\003\017v\245\235b\004\345\211\225\021[\203=\256/
    K\256\375\032\217:\351\024\327\304\342\312B\'\311\016\007\304^\2052\322\31
    6r+"))
  (propagate)
  (tag (net.espeak.method (*) (*) (*)))
  (not-before 2000-05-25_10:15:28)
  (not-after 3000-05-25_10:15:28)
  ) (signature (hash SHA-1
    "\032\356V\317\260\t\347\331\277]\'\0278\237\0301t\212#\210") (public-key
    elgamal-pkcs1 "\003\017v\245\235b\004\345\211\225\021[\203=\256/
    K\256\375\032\217:\351\024\327\304\342\312B\'\311\016\007\304^\2052\201?.U
    q\236)"))
    "\003\017`97\317`\377\'\303\347yC\337+\221$Sm\202\242\201\214w\004\352&\31
    0]U-/\272\342\301\330\034"))
```

A certificate that a service provider, a print service in this example, provides to a client that allows the client to invoke an operation, such as print, on it looks as follows (the binary data has been chopped):

```
(signed (cert
  (issuer (public-key elgamal-pkcs1
    "\003\017v\245\235b\004\345\211\225\021[\203=\256/K\256\364p\255|\335"))
  (subject (public-key "elgamal-pkcs1"
    "\003\017v\245\235b\004\345\211\225\021[\203=\256/
    K\256\311\301\273\307l\323\316r+"))
  (propagate)
  (tag (net.espeak.method PrinterServiceIntf (* set print) (*)))
  (not-before 2000-05-25_10:15:28)
  (not-after 3000-05-25_10:15:28)
  ) (signature (hash SHA-1
    "\0237\340}\310I\2638mq\207V\265\357\342\201\2702\274")
  (public-key elgamal-pkcs1
    "\003\017v\245\235b\004\345\211\225\021[\203=\256/
    K\246\374\032\020\244\004T\307\221\037\247\034\332\365\3648~\300\274\3
    62")))
```

When a message invoking an operation is received, J-ESI extracts the interface and method from it, and gets the service identifier from the information passed to the service handler by the core. From this it constructs the tag required to authorize the operation. The authorizer first checks to see if the tag is contained in the resource mask, and if it does, the operation is permitted. If the operation is not in the mask, J-ESI looks for a valid certificate (or certificates) that contain the tag needed to invoke the operation. The tag matching rules used for authorization are explained in the E-speak Architecture Specification chapter on "Access Control". The certificates checked by J-ESI are those presented by the client to establish the session

J-ESI provides service providers with means to set metadata and resource masks that can mask access control to their services. See Appendix H for details on how to set up your security environments in e-speak.

The default behavior when security is enabled is to require authorization for all operations. Service providers can create masks and associate them with service elements that they create. Masks enable Service providers to allow operations without any authorization. Any client is allowed to perform any operation that is specified in the mask for the service. If an operation is not included in a mask, only

clients that have been given certificates authorizing access are allowed to invoke the operation. The security infrastructure examines the certificates that have been presented by the client to see if they contain a tag authorizing the operation. The rules for how tags authorize operations are explained in the E-speak Architecture Specification "Access Control" chapter.

Masks

There are two types of masks: the metadata mask for metadata operations and the resource mask for resource specific operations. For example, service providers can control who can mutate the metadata of the service that they have created using the metadata masks, and they control who can invoke operations on services provided by them using the resource specific masks.

Masks are specified as tags. The basic method tag format is

```
(net.espeak.method <interface name> <method name>)
```

The tag format is explained in detail in the E-speak Architecture Specification "Access Control" chapter. In the metadata mask, the interface name is the core interface being specified, and the method name is the operation in that interface. For metadata, the interface is likely to be `ResourceManipulationInterface`, and the method name one of its methods.

In the resource mask for a J-ESI service the interface name is the fully-qualified name of the interface class. The method name is the name of the method in the interface, plus the concatenated argument types. This allows overloaded methods to be distinguished.

The metadata mask is used by the in-core metaresource when performing metadata operations. The resource mask is passed to the service handler by the core for the service handler to use when performing operations on the service itself.

The masks are completely general tags, so the mask tag itself, or any of its fields, may use the tag matching features such as sets, prefixes and ranges. The interface and method names, for example, do not have to be string literals, they can be sets or prefixes.

This tag masks method `foo` in interface `net.espeak.examples.ExampleIntf`:

```
(net.espeak.method net.espeak.examples.ExampleIntf foo)
```

This tag masks all methods beginning with foo:

```
(net.espeak.method net.espeak.examples.ExampleIntf
 (* prefix foo))
```

This tag masks methods foo and bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf
 (* set foo bar))
```

Methods with prefix foo or bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf
 (* set (* prefix foo) (* prefix bar)))
```

All methods in the interface:

```
(net.espeak.method net.espeak.examples.ExampleIntf )
```

This is equivalent to

```
(net.espeak.method net.espeak.examples.ExampleIntf (*))
```

since missing trailing elements match anything.

Methods foo in InterfaceA and bar in InterfaceB:

```
(* set (net.espeak.method InterfaceA foo)
 (net.espeak.method InterfaceB bar))
```

All methods:

```
(net.espeak.method)
```

or simply

```
(*)
```

The full form of the method tag is actually:

```
(net.espeak.method <interface name> <method name> <service>)
```

In the normal case, the service handler is only interested in its own operations, so it does not care what the service field is. Since omitting a trailing field is equivalent to giving it the value (*), we omitted this detail above.

09733027-120800

Authorizing Access

General tags can be constructed using the following method in `ESSecurityEnv`:

```
ADR createTag(String s) throws IOException
```

The `IOException` subclass `net.espeak.security.adr.ADRParseException` is thrown on a parse error. The parameter `s` is a string containing the input syntax for the tag.

Method tags can be created using

```
ADR createMethodTag(String interfaceName,
                    String methodName,
                    ADR service)
```

Clients can retrieve their current security environment from the connection. For example:

```
ESConnection conn= new ESConnection("config.file");
ESSecurityEnv secEnv = conn.getSecurityEnv();
```

For the purposes of resource masks, it is usual to use a tag containing simply (*) as the service parameter. In advanced applications, the service may want to set the service parameter to its service id, but this is not necessary.

After a mask tag has been constructed, it is used in `ESServiceElement` methods:

```
void setResourceMask(ADR tag) throws ESEException
void setMetadataMask(ADR tag) throws ESEException
```

Before a service is registered, these simply affect the local state. After registration, these set the local state and update the service metadata.

Masking can be turned on or off using `ESAuthorizer`:

```
void setMasking(Boolean x)
```

When masking is off, the resource mask is ignored by the service authorizer even if set. Setting masking off in the authorizer has no effect on the resource metadata, or the in-core metaresource handling metadata operations. Masking can be turned off completely, in the core and handler, by setting a mask to null.

An `ESAuthorizer` is associated with each `ESServiceElement`, and one can obtain the authorizer associated with an `ESServiceElement` using the `getAuthorizer()` call in `ESServiceElement`.

`ESConnection` has methods for controlling the default resource and metadata masks used when services are registered:

```
void setDefaultResourceMask(ADR mask)
ADR getDefaultResourceMask()
void setDefaultMetadataMask(ADR mask)
ADR getDefaultMetadataMask()
void setMasks(ADR metadataMask, ADR resourceMask)
```

After a default mask is set, all resources registered use it until it is changed. Unless the default masks are set explicitly, `ESConnection` uses null for them, causing authorization to be checked for all operations.

Example

In the example below, the service provider sets up a mask for the print method and a mask for the `checkStatus` method. Clients who present tags that match the print method are allowed to print and all clients are allowed to invoke the `checkStatus` method. The access control for the `checkStatus` method is disabled because of the `setResourceMask` method invocation in `ESServiceElement`.

```
public static void main(String[] args)
{
    try {
        ESConnection conn = new ESConnection("espeak.cfg");
        ESSecurityEnv sEnv = conn.getSecurityEnv();
        String m2 = "(net.espeak.method PrinterServiceIntf checkStatus)";
        ADR adr2 = sEnv.createTag(m2);
        ESServiceDescription sd = new ESServiceDescription();
        sd.addAttribute(ESConstants.SERVICE_NAME, "printer");
        sd.addAttribute("Description", "my hp printer");
        ESServiceElement se = new ESServiceElement(conn, sd);
        se.setResourceMask(adr2);
        ESAuthorizer esa = (ESAuthorizer) se.getAuthorizer();
        esa.setMasking(true);
        se.setImplementation(new PrinterServiceImpl());
        se.register();
        se.advertise();
        se.start();
    }
}
```



```

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Clients who want to invoke this print service must obtain the tags to invoke this service and install them in their security environment.

Remote Connection Manager

Using advertising service, groups and communities is the preferred way of exchanging service metadata between various cores. However, sometimes more fine grained control is required where you want to make the service available to a specific core or engine. This can be done with the aid of remote connection manager and remote service manager.

Remote connection manager has set of simple methods that allows the application programmer to connect to the specified core, to disconnect from a core and to get a list of all connections that the local core maintains with other cores. To do this, first obtain the remote connection manager object from ESConnection.

```

ESConnection conn = new ESConnection("file.prop");
ESRemoteConnectionManager connMgr =
    conn.getConnectionManager();

```

To open a connection to a remote core, the client can invoke the `openConnection()` method on the connection manager. Typically, the client specifies the address and port number on the remote machine on which the remote core is running.

```

String url = "tcp:abc.foo.com:12345";
String id = connMgr.openConnection(url);

```

Clients can close existing connections with remote cores by invoking the `closeConnection` method in the connection manager. For example:

```

connMgr.closeConnection(id);

```

where `id` is the string that represents the id of the connection that is returned by the `openConnection()` call. To get a list of connections that are currently open with the connection manager, use the following call.

```

String[] ids = connMgr.getConnections();

```

Exporting Vocabularies

Before a Service can be exported, the Vocabulary it will use must be exported. An example of this follows:

```
package tests.java.net.espeak.jesi.remote;
import net.espeak.jesi.*;
import net.espeak.infra.cci.exception.*;
public class Export
{
    public static void main(String args[]) throws Exception
    {
        ESConnection es = new ESConnection("localhost",12345,"TCP");
        System.out.println("Connected to the first core\n");
        ESRemoteServiceManager servMgr = es.getRemoteServiceManager();
        ESRemoteConnectionManager connMgr = es.getConnectionManager();
        //Creating the vocabulary.....
        ESVocabularyDescription vocabDesc = new
        ESVocabularyDescription();
        //Setting the vocabulary name
        vocabDesc.addAttribute("Name","ESR_002Vocab");
        //Adding vocabulary properties
        vocabDesc.addStringProperty("testCaseID");
        //Registering the vocabulary
        ESVocabularyElement vocabElem = new
        ESVocabularyElement(es,vocabDesc);
        ESVocabulary vocabRegister = vocabElem.register();
        System.out.println("Vocab registered");
        ESAccessor [] accList = new ESAccessor[1];
        accList[0] = ((ESAccessorHandle)vocabRegister).getAccessor();
        System.out.println("Exporting resources : " + accList[0]);
        // Connection to second core
        ESConnection connection = new
        ESConnection("localhost",12346,"TCP");
        System.out.println("Connected to the second core");
        ESVocabularyFinder vocabFind = new
        ESVocabularyFinder(connection);
        try
        {
            ESVocabulary vocabTemp = vocabFind.find( new ESQuery(
                "Name == 'ESR_002Vocab' "));
        }catch(LookupFailedException look){
            System.out.println("Lookup before Export Failed");}
        //Export
        servMgr.exportService(accList,
            connMgr.openConnection("tcp:localhost:12346"),
```


The `exportType` is an integer that is 1 if the services are to be exported by value and 2 if the services are to be exported by reference. In addition, set the boolean flag, `level` to false if the contents of a folder are to be recursively exported, and to true if only the top level of the folder is to be exported. A similar interface can be used to import resources.

```
servMgr.importService(accessor, id, importType, level);
```

To unexport an exported resource from a remote core, the service can invoke:

```
servMgr.unexportService(accessor, id);
```

To unimport an imported resource, the service provider can invoke:

```
servMgr.unimportService(accessor, id);
```

Events

This section describes the design details of the Event Service, a lightweight, extensible Service targeted at loosely coupled, distributed applications. Events provide a publish-subscribe mechanism for communication built on top of e-speak messaging.

Event Model

E-speak supports an extended form of the familiar publish-subscribe Event Model. There are four logical entities in the e-speak Event Model whose interactions are shown in Figure 14. These entities are the *Publisher*, *Listener*, *Distributor*, and *Subscriber*.

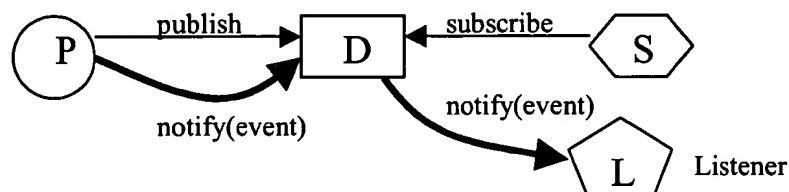


Figure 14 Interactions in the Event Model

A Publisher (marked P in the figure) is an entity that generates an Event notification message. The recipient of an Event notification is called a Listener (L). A Distributor (D) is an extension of a Listener. It receives Events and forwards them to other Listeners. A Subscriber (S) is an entity that registers interest in a particular Event with a Distributor and designates the Listener to whom Events are sent. The Subscriber and the Listener are typically the same physical entity. Similarly, it is fairly typical for a Publisher to act as a Distributor of its own Events.

The Core itself is an example of an Event Publisher. It sends Events to a trusted Client called the Core Distributor to signal state changes such as a change in a Service's attributes. The Core Distributor can then distribute these Events to interested Clients that have appropriate authority.

Interaction Sequence

Figure 15 shows a typical Event notification process where the Subscriber and Listener are folded into a single Client.

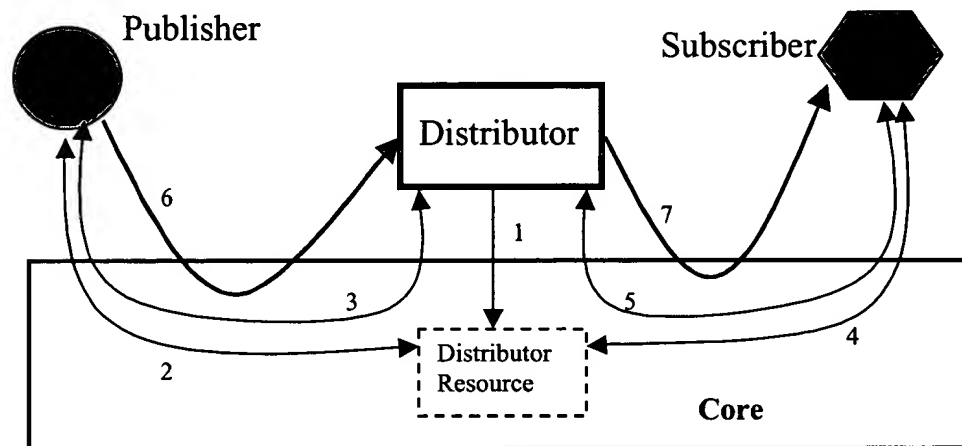


Figure 15 Typical Event notification process

The following numbers in the figure represent these steps in the process:

- 1 Distributor registers with the Core.
- 2 Publisher discovers the Distributor.
- 3 Publisher sends `publish` request to the Distributor describing the Events it will be generating.
- 4 Subscriber discovers the Distributor.
- 5 Subscriber sends `subscribe` request to a Distributor describing the Events in which it is interested.
- 6 Publisher sends the Event to the Distributor using a `notify` message.
- 7 Distributor forwards the Event to the Subscriber (also using a `notify` request).

Subscribing to Events

The Event APIs provide simple mechanism by which Clients can express interest in various Events and handle them. The following code shows how a printer Client subscribes to outofpaper and paperjam Events and handle them subsequently.

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new ESConnection(propertyFileName);
....
....
ESSubscriber printerEventSubscriber =
    new ESSubscriber(coreConnection);
printerEventSubscriber.addEvent
    ("hp.headoffice.firstfloor.printer.outofpaper");
printerEventSubscriber.addEvent(
    ("hp.headoffice.firstfloor.printer.paperjam");
printerEventSubscriber.setImplementation
    (new PrinterEventHandler());
ResultSet rs = printerEventSubscriber.subscribe();

public class PrintEventHandler implements ESListnerIntf
{
    ...
    ...
    public String notifySync(Event evt)
    {
        System.out.println(evt.getPayload());
        return "Notified" ;
    }

    public void notify(Event evt)
    {
        System.out.println(evt.getPayload());
    }
}
```

The event subscriber, after connecting to the e-speak core, creates an instance of ESSubscriber and expresses interest in certain event using addEvent call. Furthermore, the subscriber sets the handler for the Events in which it is interested using setImplementation(). The handler should implement the ESListenerIntf. Then the subscriber invokes subscribe() to register with the existing distributors in the community. The community is set using setCommunity() call in ESServiceContext. The ResultSet obtained as a

result of `subscribe()` call contains success or failure of subscription with different distributors. The subscriber can subscribe to e-speak service events in a similar manner. The subscriber can also subscribe to e-speak core events using `ESCoreSubscriber` class. The list of e-speak service and core events are mentioned towards the end of this section.

The following code gives a simple example of how to subscribe to service events, for e.g., `service.create`. This subscribes the user to any service creation events in the community.

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new ESConnection(propertyFileName);
....
....
ESSubscriber serviceCreateSubscriber =
    new ESSubscriber(coreConnection);
serviceCreateSubscriber.addEvent
    ("service.create");
serviceCreateSubscriber.setImplementation
    (new PrinterEventHandler());
ResultSet rs = serviceCreateSubscriber.subscribe();
```

Publishing Events

Publishing events is done using the `ESPublisher` class. The usage is similar to that of subscribing to events.

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new ESConnection(propertyFileName);
....
....
ESPublisher printerEventPublisher = new
    ESPublisher(coreConnection);
printerEventPublisher.addEvent
    ("hp.headoffice.firstfloor.printer.outofpaper");
printerEventPublisher.addEvent
    ("hp.headoffice.firstfloor.printer.paperjam");
printerEventPublisher.publish();
....
....
Event outofpaperEvent =
    new Event("hp.headoffice.firstfloor.printer.outofpaper");
outofpaperEvent.setPayload("printer foo out of A4 paper");
printerEventPublisher.sendNotify(outofpaperEvent);
```


Just as the subscriber expresses interest in receiving events, the publisher expresses interest in sending events. The publisher does this by instantiating the `ESPublisher` class, adding events of interest using `addEvent` and then calling `publish()`. `publish()` does not send an event to the consumer, rather it just expresses intent to publish those events at a later point of time. Actual publishing of events takes place when the publisher calls `sendNotify()` after constructing an `Event` object. There is a default publisher available with the `ESConnection` that can be obtained using the `getDefaultPublisher()` call. This is used for publishing only the e-speak service events. The list of service events is given at the end of this section.

Distributing Events

So far we have talked about a simple subscriber and publisher assuming that there is already a distributor is available for the outofpaper and paperjam events. In case no such distributor is available, the publisher of the events can write a simple `Event` distributor as follows.

```
String propertyFileName = new String("/users/connection.prop");
ESConnection coreConnection = new ESConnection(propertyFileName);
....
....
ESDistributor printerEventDistributor = new
    ESDistributor(coreConnection);
printerEventDistributor.addEvent
    ("hp.headoffice.firstfloor.printer.outofpaper");
printerEventDistributor.addEvent (
    ("hp.headoffice.firstfloor.printer.paperjam");
printerEventDistributor.start();
....
....
printerEventDistributor.shutdown();
```

The distributor creates an instance of `ESDistributor` and adds events which the distributor intends to distribute using the `addEvent()` call. The distributor then starts using the `start()` call. The distributor can be stopped using `shutdown()` call. There are some pre-existing `Event` distributors bundled with the e-speak core. These are started along with the e-speak core. They are the service distributor and the core distributor. These distribute the service and core events listed at the end of

this section. It is possible to subscribe to core events distributed by the `ESCoreDistributor`. It is possible also to subscribe and to publish service events that are distributed by the `ESServiceDistributor`.

List of Service Events

This section lists the Service Events generated by e-speak.

<code>service.create</code>	This Event is generated by e-speak Service interface when a Service is created.
<code>service.mutate</code>	This Event is generated when a Service's attributes are mutated.
<code>service.delete</code>	This Event is generated on deletion of a Service in the e-speak Service interface.
<code>service.access</code>	This Event is generated whenever the <code>ESAccessor</code> of a Service is used.
<code>service.pause</code>	A Service can voluntarily generate this Event for temporary pause of its Services.
<code>service.resume</code>	A Service can voluntarily generate this Event on resumption of its Services.
<code>service.genericinf</code> o	A Service sends out generic information about itself through this Event type.

List of Core-Generated Events

This section lists the Events generated by the e-speak Core.

<code>core.mutate.NameFrameInterface.3</code>	Bind a Resource to a new name in an existing Name Frame.
<code>core.mutate.NameFrameInterface.4</code>	Rebind an existing name in a Name Frame to a new Resource.
<code>core.mutate.NameFrameInterface.5</code>	Unbind a name from a Resource.

<code>core.mutate.NameFrameInterface.6</code>	Copy a binding from one Name Frame to another.
<code>core.mutate.NameFrameInterface.7</code>	Add a binding to a Resource to an existing name.
<code>core.mutate.NameFrameInterface.8</code>	Remove a binding to a Resource from an existing name.
<code>core.mutate.VocabularyInterface.3</code>	Modify the attribute set of a Vocabulary.
<code>core.mutate.ResourceFactoryInterface.1</code>	Register a new Resource.
<code>core.mutate.ImporterExporterInterface.1</code>	Import a new Resource.
<code>core.mutate.ImporterExporterInterface.5</code>	Update an imported Resource.
<code>core.mutate.ResourceManipulationInterface.1</code>	Unregister a Resource.
<code>core.mutate.ResourceManipulationInterface.3</code>	Modify the owner of a Resource.
<code>core.mutate.ResourceManipulationInterface.5</code>	Modify the handler of a Resource.
<code>core.mutate.ResourceManipulationInterface.9</code>	Modify a public RSD of the Resource.
<code>core.mutate.ResourceManipulationInterface.11</code>	Modify a private RSD of the Resource.
<code>core.mutate.ResourceManipulationInterface.13</code>	Modify the attributes of the Resource.
<code>core.mutate.ResourceManipulationInterface.23</code>	Modify the export-type of the Resource.
<code>core.failure.invalid_parameter</code>	Pass an invalid parameter.
<code>core.failure.null_parameter</code>	Pass a null parameter.
<code>core.failure.invalid_value</code>	Receive an invalid value.
<code>core.failure.invalid_type</code>	Receive an invalid type.
<code>core.failure.out_of_order</code>	Out of order.
<code>core.failure.Core_panic</code>	Core has an irretrievable exception.

SECRET

Appendix A Thread-Safe Programming: ESThread

There are two aspects to threading when programming with J-SEI.

- The threads that the service handler uses to handle requests to the service
- The threads in the client program that may share a connection to the core.

When service providers create services, they create service handlers. These service handlers can be set up so that there are many threads that handle requests to this service. Essentially, each service handler comes with a `ESThreadPoolManager` that manages the pool of threads for servicing requests to this service. Furthermore, there are two main policies that can be used by the service provider:

- **Thread per request:** When the configuration indicates that the service provider wants to spawn off a thread per request, the `ESThreadPoolManager` creates a new thread to handle every request for the service. This thread runs to completion at the end of the request.
- **Fixed thread pool:** In this case, the thread pool manager has a fixed pool of threads that depends on the number of threads that the client has indicated in the `ESServiceHandler`. The client can set the number of threads she wants by using the `setNumThreads(int num)` in the `ESServiceHandler`.

Client applications can be multithreaded. A Client can choose to create a connection to the Core and then create multiple threads (using the native Java threads) that use this connection simultaneously.

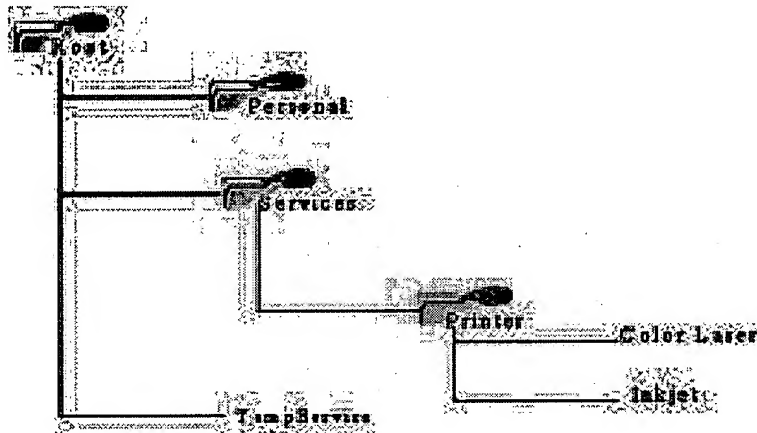


Figure 16 An example of threads

Because these threads share the same connection, any state stored in `ESConnection` is shared across all these threads. In certain situations, this is undesirable when the threads need to operate independently of each other. `ESThread` is used in these situations.

An example of where this problem can occur is the use of the `setCurrentFolder(...)` API, to set the current folder in which new bindings are created. If multiple threads need to change the working directory without affecting other threads, then these threads should be created using an `ESThread()` call, which clones the `ESConnection` state on creation of the thread.

Just like Java Threads, there are two ways of using the J-ESI threads.

- Application developer creates a Java file which implements `ESRunnable` (counterpart of Java `Runnable`) and passes this to the constructor of `ESThread` (counterpart of Java `Thread`). The application developer does this if his Java file already extends some other Java class and hence could not extend `ESThread` directly. This is the suggested and preferred way of using `ESThreads`.

```
ESThread thread1 = new ESThread(esrunnable);
thread1.start();
```

Application developers can also extend ESThread directly. An example is as shown below.

```
public ApplThread extends ESThread
{
    public ApplThread(ESConnection coreConnection)
    {
        super(coreConnection);
    }

    public void run()
    {
        ESConnection clonedConnection =
            super.getConnection();
        ....
        ....
        clonedConnection.getServiceContext().
            setCurrentFolder(...);
        ....
        ....
    }
}
```

For example, the following code fragment creates two threads that operate on the same connection:

```
public class ThreadTest
{
    public static void main(String args[])
        throws ESInvocationException, ESLibException,
        InterruptedException, IOException
    {
        String propertyFileName = new String("/users/connection.prop");
        ESConnection coreConnection = new ESConnection(propertyFileName);
        ApplThread thread1 = new ApplThread(coreConnection);
        ApplThread thread2 = new ApplThread(coreConnection);
        thread1.start();
        thread2.start();
    }
}
```

0973027-120800
008027-120800

Appendix B Messaging Classes

E-speak Clients discover Services and interoperate with them using a remote object model or using Events. In addition to these approaches, e-speak supports a purely messaging interaction model with the `ESServiceMessenger` class.

Clients send and receive messages using synchronous or asynchronous messages to other Services represented by the `ESAccessor` associated with the Service.

```
ESServiceMessenger messenger =  
    new ESServiceMessenger (coreConnection);
```

The messenger can be used to send a payload object synchronously or asynchronously to any other Service. To send an object, use the following code:

```
...  
Object payload = new Object();  
ESService myService = finder.find(query);  
ESAccessor serviceAccessor = ((ESAccessorHandle) myService).  
    getAccessor();  
Object retValue = messenger.sendSynchronous  
    (serviceAccessor, (Object) payload);
```

or (to send asynchronously):

```
ESMessage msg = messenger.sendASynchronous  
    (serviceAccessor, (Object) payload);
```

When an asynchronous message is sent, a message ID is returned in the form of an `ESMessage`. The service sending an asynchronous message may expect a notification to be returned. It can wait for a reply using the `wait` method as follows:

```
messenger.wait(msg);
```

where `msg` is the return value of the asynchronous send.

```
ESMessage msg = messenger.receive(se);
```

On receipt of a message, a Service can extract its payload, operate on it. To extract the payload from the `ESMessage`, use the following method:

On operating on the payload, the service can reply to a message by invoking the `reply()` method on it.

```
Object repObject = new Object();
msg.reply(repObject);
```

Appendix C IDL Compiler

The purpose of ESIDL compiler is to prepare the code that enables programmer of client code to invoke services as if they were running in the same process space. Such code includes stub, registering into message registry and serialization of objects.

ESIDL compiler can create new code and check existing code for conformance. Both e-speak serialization and Java serialization are supported.

Generating code

ESIDL compiler uses as input description of services, types, and exceptions and creates support files needed by user to invoke service through e-speak.

There are two kinds of description:

- services are described in **service description files**
- types and exceptions are described in **type description files**.

There are several kinds of support files:

- **interface files** that define service interfaces
- **stub files** that represent service on client side
- **message registry files** that are used to register classes with e-speak messaging layer
- **class files** that implement data types and exceptions

Data type description can be in an ESIDL file (extension “.esidl”) or in a Java file (extension “.java”). Service description can be only in an ESIDL file. ESIDL files are checked and new code is generated from them. If files of both types exist, then the ESIDL file is used as input.

Processing Java files is new to ESIDL compiler.

Input Files

Service Description

Service description includes information about an e-speak service. For ESIDL compiler every parameter it receives on the command line denotes an e-speak service. This is a change from the previous version where all service files and data types had to be listed. Interface file, stub file, and message registry file are generated for every e-speak service.

```
Service.esidl & ServiceIntf.java, ServiceStub.java,  
ServiceMessageRegistry.java
```

Type Description

Type description includes information about:

- types of service attributes
- types returned by service methods
- parameters to service methods
- exceptions thrown by service methods

Type description files are not listed as command line parameters to the compiler. They are searched for using import statements in the same manner as javac compiler searches for the source files (CLASSPATH environment variable or -classpath option).

Type.esidl → generated Type.java
Type.java → checked Type.java

Output Files

Interface File

Interface file includes interface that the service implements. The interface extends ESService.

Stub File

Stub file includes code for:

- serialization of service stub object
- rerouting of service method invocations through core
- calling messaging registry initialization (in message registry file) for return types, parameter types and exception types used in the service

Message Registry File

Message registry file includes code for registering types (for a list, see Type Description chapter) with MessageRegistry.

Type File

Type file is generated from type description. Serialization code is generated (for .esidl files) or checked (for .java files). E-speak serialization is default if none is specified (in declaration of type). Otherwise the specified serialization is used.

Do not list all data type and exception files as the compiler parameters as it was required by the previous version of the compiler.

where:

Semicolon separated list of directories where compiler searches for description files. "." (dot) is used for current directory. If classpath is not specified, environment variable CLASSPATH is used.

Instructs the compiler to print out information about created files.

Requirements have changed from the previous version and are now much less strict.

Service description is admissible if:

- it is declared `public` AND
- all methods it declares are admissible

A user-defined abstract class or a user-defined exception is admissible if:

- it is declared `public` AND
- its parent type is either `java.lang.Object` or another admissible user-defined data type

A service method is admissible if:

- return type and parameter types are admissible
- exceptions thrown are admissible

Description in Java file

A **user-defined data type** (a Java class) must conform to these additional restrictions:

- implements `ESSerializable` or `java.io.Serializable` interface

A **user-defined exception** (a Java class) must conform to these additional restrictions

- it must extend `ESServiceException`

A **service method** is admissible if it throws `ESInvocationException`

00333027-120800

09733027-120800

Appendix D Interceptors

J-ESI provides simple mechanisms for service providers and clients to monitor accesses to their services. Service providers can use these mechanisms to either generate management events such as billing events, create an access log, or provide mechanisms for performing load balancing, redirecting requests should a particular server be unavailable, etc. Clients can use interceptors for adding or removing parameters from requests or even implement a secure invocation interceptor that finds an available service with the required method and invoke it.

In essence, each `ESServiceElement` object contains an `ESInterceptorControl` object that controls the list of interceptors associated with the service element. The interceptor objects are instances of classes that extend the abstract `ESInterceptor` class. The `ESInterceptorControl` object therefore provides methods provides methods to add and remove interceptors from the `ESServiceElement`. When a message arrives for the service represented by the `ESServiceElement`, the message is passed through each interceptor that the service provider has associated with the `ESServiceElement`.

Interceptors are classified based on whether they are terminal or not. A terminal interceptor is an instance of `ESTerminalInterceptor`, and represents an actual invocation to the service. There is a single terminal Therefore, the `ESInterceptorControl` that is associated with any `ESServiceElement`, comes with a default `ESTerminalInterceptor`. However, the service provider can change that terminal interceptor with any other terminal interceptor that she writes.

The service provider must go through the following steps in order to set up an interceptor:

- Define an interceptor class that extends `ESInterceptor` or `ESTerminalInterceptor`. The service provider must implement the `invoke (ESRequest req)` method in the extended class. Furthermore, if the interceptor being defined is an extension of

ESIceptor, she has to make a `invokeNext()` call in the implementation of the `invoke` method. If this is not done, the interceptors that are added after this interceptor are not invoked.

- The service provider also must implement the `initialize(Object params)` method in the interceptor class. This method is invoked when the interceptor is added to the `ESIceptorControl` and can be used to initialize the state of the interceptor.
- When the service provider creates a `ESServiceElement` that represents the service, she makes an `addIceptor(ESIceptor icp, Object params)` that adds the interceptor to the interceptor control object associated with the service element.

The service provider is free to override other methods in `ESIceptor`, but the `invoke`, and `initialize` methods must be implemented.

We now present a simple example that shows using the interceptors. Consider the print service example from the previous sections. Suppose the service provider wants to:

- balance the load among multiple print service implementations
- generate information about each access to the print service

This is accomplished by the following interceptors: the `LoadBalancingInterceptor`, and the `PrintingInterceptor`

```
public class LoadBalancingInterceptor extends ESIceptor {
    // Array of backup services
    private ESService[] services;
    private ESService thisService;

    // If the service reaches this load backup services are invoked
    private int LIMIT_LOAD = 5;
    private String PRINT = "print";

    public boolean invoke(ESRequest req) {
        System.out.println("LoadBalancingInterceptor enter invoke");
        if (req.getMethodName().equals(PRINT)) {
            try {
                ESRequest request = new ESRequest();
                // check services load
                int load = ((PrintServiceIntf)thisService).getLoad();
```

Interceptors

```
        if (load > LIMIT_LOAD){
            // the load is above limit load, invoke backup service
            int minLoadService = getLowestLoadService();
            PrintServiceIntf ps = (PrintServiceIntf)services[minLoadService];
            req.setReturnValue(ps.print((String)req.getParamValue("arg0")));
            invokeNext(req);
        }else{
            invokeNext(req);
        }
    }catch(Exception e){
        return false;
    }
    }else{
        try{
            invokeNext(req);
        }catch(Exception e){
        }
    }
    }
    return true;
}

public void initialize(Object param){
    // .. implementation that initializes the services array, etc.
    // from information in param
}

private int getLowestLoadService(){
    //... implementation
}

//This is how the interceptor code looks like

public class PrintingInterceptor extends ESInterceptor {
    int cnt=0;
    public boolean invoke(ESRequest req) throws ESInvocationException{
        if(req.getMethodName().toString().equals("print")){
            cnt++;
            writeCountToLogFile();
        }
        try{
            invokeNext(req);
        }catch(Exception e){
        }
    }
}
```

```

        return true;
    }
    private void writeCountToLogfile(){
        // implementation
    }

    public void initialize (Object param){
    }
}

```

Now that we have defined the two interceptors, the print server, looks as follows:

```

public class PrintServer
{
    public static void main(String [] args)
    {
        try {
            String propertyFileName = new String("/users/connection.prop");
            ESConnection coreConnection = new ESConnection(propertyFileName);
            ESServiceDescription printDescription =
                new ESServiceDescription();
            printDescription.addAttribute("Name","printer");
            ESServiceElement printElement =
                new ESServiceElement(coreConnection, printDescription);
            printElement.setImplementation(new PrinterServiceImpl());
            printElement.register();
            printElement.addIceptor(new PrintingInterceptor(),"HP Lobby 49 U
printer");
            LBIParams lbiParams = ../ initialize object expected by
//initialize method of LoadBalancingInterceptor
            printElement.addIceptor(new LoadBalancingInterceptor(),
                lbiParams);
            printElement.start();
            System.out.println("Started printer Service ");
        }
        catch (Exception e){
            // handle the exception
        }
    }
}

```

In the above example, the printing interceptor is invoked before the loadbalancing interceptor on any request. If the service provider wants the order to be reversed, reverse the order of adding the interceptors to the service element.

Appendix E Account Manager

The Account Manager allows administrators to create and manage accounts in e-speak. This means that administrators can create accounts, grant or change permissions for accounts, and occasionally delete accounts. In the current release, the Account Manager cannot grant, change or revoke account permissions, but this will be added in a future release.

Programming Model

The Account Manager module provides three basic abstractions, the Account Manager, the Account Profile, and the Account Description. The Account Manager allows basic account management functions, including creating and deleting accounts, authenticating users, and retrieving lists of accounts. Each account created by the account manager has an account profile associated with it, which is required for authentication of the account. To change the description associated with the account profile, a profile description with a valid (non-null) vocabulary must be supplied, together with the attributes to be added to the description.

Profile Description

The Profile Description describes attributes of the Account Manager. Since the default Account Manager vocabulary does not define a set of attributes that are expected, the user must provide a vocabulary that describes the attributes which will be included in this profile description, so that it can be added to the existing description of the account profile. In the account manager module, this abstraction is represented by the `ESProfileDescription` class.

Account Profile

The Account Profile contains the authentication information for the account. It is initialized with just the account name and password phrase and can be modified by adding a Profile Description to it. In the account manager module, this abstraction is represented by the `ESAccountProfile` class.

Account Manager

The Account Manager allows the service administrator to perform administrative tasks such as creating and deleting accounts, retrieving lists of all accounts, and adding descriptions to existing accounts. The service administrator's main interaction to the Account Manager module is through the Account Manager abstraction itself. In the account manager module, this abstraction is represented by the `ESAccountManager` class.

In this release, there are two default accounts defined in e-speak's core Account Manager: the admin account, and the guest account. The username, passphrase pair for the admin account is "admin, admin", and the corresponding pair for the guest account is "guest, guest".

A Simple Example

Consider an example where a service administrator decides to create an account temporarily, change its description, and then delete the account. At present, only an administrator or someone presenting the credentials of the account itself can delete an account. In a future release, there will be functionality allowing the granting of permissions to new accounts, and it is probable that some new accounts will be granted permissions to delete at least some subset of the existing accounts.

First, to create the account, the service administrator should open a connection to an e-speak core. (The core is already running). After a connection has been established, its `getAccountManager()` method can be used to retrieve the active `ESAccountManager`. The `ESAccountManager` can now be used by the administrator to perform normal account maintenance activities.

```
String propertyFileName = new String("/users/connection.prop");
ESConnection esconn = new ESConnection(propertyFileName);
ESAccountManager acctMgr = esconn.getAccountManager();
```

Now that we have retrieved the ESAccountManager, the administrator can create an account based on an account name. First, an ESAccountProfile must be created using that account name. The newly created ESAccountProfile is passed in as the associated profile for the account to be created.

```
ESAccountProfile acctProf = new ESAccountProfile("testAccount");
String acct = acctMgr.createAccount(acctProf);
```

The account just created can be referred to by the java.lang.String returned from the creation. The administrator can now add a description to the account. To do this, he/she must first ensure that there is a valid vocabulary associated with it, then must create an ESProfileDescription and add attributes to it that describe this ESAccountProfile.

```
ESVocabularyDescription vocDesc = new
ESVocabularyDescription();
vocDesc.addAttribute(ESConstants.SERVICE_NAME, "testVocab");
vocDesc.addStringProperty( "manufacturer" );
vocDesc.addStringProperty( "model" );
vocDesc.addStringProperty( "year" );
ESVocabularyElement vc = new ESVocabularyElement(esconn,
                                                    vocDesc );

ESProfileDescription desc = null;
ESAttribute [] atts = null;
ESVocabulary vocab = null;

try {
    vocab = vc.register();
} catch (NameCollisionException nce) {
    nce.printStackTrace();
} catch (ESLibException esle) {
    esle.printStackTrace();
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}

desc = new ESProfileDescription(vocab);
desc.addAttribute("manufacturer", "Honda");
desc.addAttribute("model", "Civic");
desc.addAttribute("year", "2000");
```

```

try{
    acctMgr.addDescription(acctProf, acct, desc);
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}

```

Now that there is an active account, with an account description, the properties just described in the ESProfileDescription can be used to find this account again. To do this, an ESQuery is constructed, describing the value desired for the attribute in question, and this query is used as the argument to a find() operation in an ESServiceFinder. This retrieves the list of services (accounts, in this case) that satisfy the requirements of the ESQuery, and then the service administrator can switch to this account using the switchAccount() method of the active ESConnection.

```

ESServiceFinder finder = new ESServiceFinder(esconn);
try {
    ESQuery query =
        new ESQuery(vocab,
            "(manufacturer == 'Honda') or (year == '2000')");
    ESService[] serviceList = finder.find(query);

    ESAccessor accessor = ((ESAccessorHandle)
serviceList[i]).
                                getAccessor();

    esconn.switchAccount(accessor);
    esconn.init();
} catch (LookupFailedException lfe) {
    lfe.printStackTrace();
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}

```

NOTE: The core finder assumes that a single char in single quotes is of type char and multiple characters in single quotes are of type String. If use a single char constraint for an attribute of type String the single char should be enclosed in double quotes and then it is interpreted by the core as string type.


```

Properties prop = new Properties(System.getProperties);
prop.put("manufacturer", "Honda");
try {
    String propertyFileName = new String("/users/
connection.prop");
    ESConnection testConn = new ESConnection(propertyFileName);
} catch (ESInvocationException esie) {
    esie.printStackTrace();
} catch (ESLibException esle) {
    esle.printStackTrace();
}

```

If there is a need to retrieve the `ESAccountProfile` of an account, for example, when an account has been retrieved using one of the methods described above, the `getAccountProfile()` method of the `ESAccountManager` can be used.

```
try {
    ESAccountProfile retProf = acctMgr.getAccountProfile(
                                                acctProf, acct);
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}
```

The administrator can also set the `ESAccountProfile` for a given account, to allow, for example, changing of the password for the account. As in the previous example, a valid account profile is required in order to call the `setAccountProfile()` method on an account.

Developer Release X.03.03.00, September 2000

```
try {
    ESAccountProfile retProf = acctMgr.setAccountProfile(
        acctProf, newProf);
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}
```

The `getAllAccounts()` method can be used to retrieve a list of all the existing accounts. This list can, for example, be useful for checking whether a particular account name is already used. This method can be called with no authentication required for the caller.

```
try {
    String[] accounts = acctMgr.getAllAccounts();
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}
```

Finally, to delete an account, the caller must again provide a valid `ESAccountProfile`, as well as the name of the account to be deleted. The account is only deleted if the provided `ESAccountProfile` can be successfully authenticated.

```
try {
    acctMgr.deleteAccount(acctProf, acct);
} catch (ESInvocationException esie) {
    esie.printStackTrace();
}
```

09733027 120300

053067 1000000

153

Access control to services is done by exchanging digitally signed certificates as a part of the SLS protocol providing secure sessions. These certificates act like “tickets” that grant entities with authorization to access and make use of services. Certificates are signed by issuing entities (or Principals) and are issued to subject principals who may use them. These certificates can also be chained together (using *delegation*) to give composite authorizations.

Refer to the E-speak Architecture Specification chapter on “Access Control” for further details concerning the security model.

PSE's and Certificates

A Private Secure Environment (PSE) represents a keystore containing public/private key pairs. Each principal e-speak entity needs to have their own set of keys and thus needs to store them securely within a PSE. The PSE itself can be stored as a binary file in your local file system. This data is encrypted and a passphrase is required to lock/unlock the data it contains.

The PSE is responsible for generating it's own key pairs—in particular, it has been designed so that private keys should never be exposed¹.

The other main function of a PSE involves validating and signing certificates. Validating a certificate involves checking the signature of the certificate using the issuers public-key (embedded within the certificate). Signing a certificate involves using a private key held within a PSE to create a digital signature, based upon a message digest of the certificate data.

PSE Manager

The PSE Manager is a GUI tool that supports these basic tasks:

- 1 Creating a new PSE and saving it as a binary file.
This involves selecting a passphrase that is used as an encryption/decryption key. It is important to keep this passphrase information secure—anyone capturing your PSE can *perfectly* masquerade as you and access everything that you can access. Also, losing or forgetting your passphrase means that you are
- 1 There is currently a method that can expose private keys – but this is only temporarily present to accommodate a deprecated internal interface that soon it is unnecessary to support.

unable to unlock or access your own PSE. For automatic operation, the PSE passphrase can be kept in a pass file stored on a floppy disk etc. There is a configuration option for this.

- 2 Creating new keys-pairs.
The PSE Manager can create new key-pairs, each of which are given a symbol label. These labels can then be used when constructing certificates. Note that this labels are referred to as roles in `espeak.cfg` and some of the security code.
- 3 Creating and editing certificates.
Attribute certificates typically contain information about the issuer, the subject, what is being authorized and the validity period. For convenience, the PSE's symbolic labels (or *roles*) for keys can be used to refer to known keys when constructing certificates—thus avoiding tedious and error-prone data entry of key information.
- 4 Validating and signing certificates as described above.

PSE data can be saved to binary files (using a passphrase for the encryption key) and certificates can be saved to text files etc.

For further information on the PSE Manager, refer to the PSE Manager user documentation.

Bootstrap process for testing

The bootstrap process *for testing purposes* is as below. When writing and deploying secure applications, refer to the J-ESI documentation and the E-speak Architecture Specification.

- 1 Use the PSE Manager GUI tool to do the following:
 - a Generate a keystore object (i.e. a Private Secure Environment) and is typically called `securestore.txt`. This is presently shared by all participants—the core, the client and the service. Therefore, this configuration is *not* distributed.

- b Each participant has their own key-pairs. The current simple approach is to generate three different key-pairs, one for each participant, with the following labels : `client`, `core`, and `service`, all within the same PSE.
- c Generate a basic attribute certificate, one for each pair of distinct participants (i.e. `client` as issuer, `core` as subject and so on for all distinct combinations) which gives each participant *arbitrary* permission to perform operations. The PSE Manager can be used to conveniently generate these attribute certificates—it has access to all the keys that were generated. The PSE labels associated with the key-pairs can be used to refer to the keys within the certificates for convenience.
- d After it is generated, the important thing is that the certificate must be *issued*—this means it is *signed* by the Core itself. Thus, the certificate is issued by the Core, and having the Core again as its subject, with an all-powerful e-speak tag attribute:

```
(net.espeak.method (*) (*))
```

Again, the PSE Manager can perform this function of signing these certificate by any one of the participants.

Significantly, the PSE Manager GUI tool is generally standalone and does not need any prior configuration, i.e., it does not require any configuration before it can be used.

- 2 To operate the core with security turned on, a security configuration file needs to be correctly loaded containing the appropriate attributes. The configuration file is more fully explained in the following sections. But a high-level snapshot goes as follows:
 - a The configuration file is like a Java properties file and is typically named `espeak.cfg`. It is searched for in the current directory, the user's home directory or on the Java CLASSPATH.
 - b A very simple `espeak.cfg` file is shown on the next page.

```
!=====
! $Id: espeak.cfg,v 1.1.2.1.4.10 2000/05/16 07:17:42 ks Exp $
!=====
! E-speak security properties file.
```

```

!=====
!-----
! Security properties.
!-----
! Master flag controlling whether security is on or off.
net.espeak.security.activate=on

! Set a property prefix.
@prefix=net.espeak.security

! Default name of the keystore file
.pse.storefile=securestore.txt

! Gui mode runs a dialog for the passphrase.
!.pse.mode=gui
! Passphrase mode looks for the passphrase property.
.pse.mode = passphrase
! Passfile mode looks for a file containing the passphrase property.
!.pse.mode = passfile

! Define the passphrase.
.pse.passphrase = default passphrase

! Define the default role (i.e. the default PSE key label).
!.pse.role = client

```

The following section discusses configuration files in more detail.

Configuration files

The default configuration file is `espeak.cfg`. The file is looked for in the following places:

- config directory under e-speak home as defined by property 'espeak_home' directory specified by property 'net.espeak.util.config.file', or current directory (from system property 'user.dir') if the property is not set directory specified by 'user.home' system property as a system resource from the classpath Java system properties can be set on the java command line using the syntax -Dproperty=value.

- The name of the file to look for can be specified using the 'net.espeak.util.config.file' property. The file defined by the property 'net.espeak.util.config.master' is always loaded on top of all other files, if specified. The default for this property is null.
- All files found are loaded, in reverse order, with files found earlier being merged on top of properties from files found later. The format of the files is java properties file format, with the following additions.

```
@prefix=<prefix>
```

This sets a property prefix to apply to properties starting with a dot. For example:

```
@prefix=net.espeak.security
.pse.mode = passphrase
```

results in net.espeak.security.pse.passphrase being set to 'passphrase'. After it is set, a prefix remains in force until changed or set null.

```
@mode=<mode>
```

If the <mode> is "override" (default) the values found in this file is used and all previous values are ignored. After the espeak.cfg parser encounters a file with mode set to "override", no more files are parsed. If the mode is "merge" espeak.cfg files are combined, if two files specify values for the same property, the value in the last file to be parsed is used.

The name of the configuration file to look for can be set using the system property

```
net.espeak.util.config.file
```

which has the default value espeak.cfg.

If the system property

```
net.espeak.util.config.master
```

is set, the file of that name is loaded on top of all other files found.

The configuration is got by calling

```
ConfigIntf Config.getInstance()
```

which returns a reference to a static instance of the default configuration. Other files can be loaded directly if wanted, see util.Config for the API. Single property files can be loaded using ConfigProps.

09733027.120800

Property file syntax

A java properties file contains property names and definitions. The name is separated from the definition by '='. Spaces before the property name and around the = are ignored. The value of the property extends to the end of line, and includes trailing spaces. Long property values can be broken across lines using \ to escape new lines. The characters ! and # introduce end-of line comments. The character : may be used as an alternative to =.

Property conversion

The class util.Convert provides methods to convert property strings to and from common types. The types int, boolean, and long are supported. The duration converters accept times in the format 12h3m1.001s and convert them to longs in milliseconds. Any zero component of a time can be omitted, and spaces may be included. A zero time can be stated as 0s.

The boolean converter accepts on, true, yes for true and off, false, no for false, regardless of case.

Argument specifications

The mapping of argument switches to properties can be defined using util.ArgSpec. This provides methods to process command-line arguments and map them onto properties in a configuration.

Security properties

The following are the properties supported by the security code.

- Master flag controlling security: `net.espeak.security.activate`, boolean, default off. If this property converts to true, security is activated.
- Property `net.espeak.security.connectOnContact`, default off, controls whether secure sessions are established with newly encountered resources. When it is off, sessions are not established unless required (by `SessionRequiredException`) or created explicitly.

- PSE mode: `net.espeak.security.pse.mode`. Values: `gui`, `passphrase`, `passfile`. Default `gui`. If the mode is `gui`, a dialog is used to get the PSE passphrase. If the mode is `passphrase` the property `net.espeak.security.pse.passphrase` is used to get the passphrase (default `null`). If the mode is `passfile` the property `net.espeak.security.pse.passfile` (default `passfile.txt`) is used to get the name of a file which must contain a `net.espeak.security.pse.passphrase` property defining the passphrase.
- PSE key file: `net.espeak.security.pse.storefile`, default `securestore.txt`. Defines the name of the file containing public-private key pairs.
- PSE role: `net.espeak.security.pse.role`, default `client`. Define the default role (symbolic PSE key name).
- PSE file protection mode:
`net.espeak.security.pse.OSfileprotection`, default `true`. This property specifies whether local OS file protection should be applied and is supplied purely as an aid for testing purposes. For full security protection, this option should not be `false`.
- Certificate file suffix: `net.espeak.security.pse.certfile`, default `certs.adr`. The value of this property is appended to the role name to get the name of the certificate file to load. If the role is `client` the certificate file is `clientcerts.adr` for example.
- ACL file suffix: `net.espeak.security.pse.aclfile`, default `acl.adr`. The value of this property is appended to the role name to get the name of the ACL file (trust assumptions) to load. If the role is `client` the ACL file is `clientacl.adr` for example.
- Cipher suites: `net.espeak.security.cipherSuites`. The value is a list of cipher suites in ADR syntax. The default is to use `hmac`, `sha-1`, and 128-bit `blowfish`.
- Public key: `net.espeak.security.pse.publicKeyAlgorithm` using the values `ELGAMAL` and `RSA`. The default is `ELGAMAL`. Which one is being used can be found using `net.espeak.security.util.PublicKeyLib`. Call `public static String getPublicKeyAlgorithm()` to find out.

003027" 120000

We support two public key algorithms: RSA and El Gamal. The entire system must use one or the other, mixing is not supported. Public key classes are not loaded statically—they are loaded dynamically based on the configured algorithm.

When El Gamal is configured, support for RSA is not be loaded, and if RSA data is encountered a `NoSuchAlgorithmException` is thrown. Which algorithm is used is defined by the property.

Example espeak.cfg file

```
!=====
! E-speak security properties file.
!=====
! Master flag controlling security.

net.espeak.security.activate=on
user.name="John Doe"

! Example time value.

foo.timeout = 12h 3m .0001s
! Set a property prefix.

@prefix=net.espeak.security
! Gui mode runs a dialog for the passphrase.

!.pse.mode=gui
! Passphrase mode looks for the passphrase property.

.pse.mode = passphrase
! Passfile mode looks for a file containing the passphrase property.

!.pse.mode = passfile
! Define the passphrase.

.pse.passphrase = default passphrase
! Define the default role (PSE key name).
!.pse.role = foo
```

00802F" 20EE460

Appendix G Firewall Traversal

This chapter discusses firewall traversal using e-speak. We present how services can be accessed using e-speak while sitting behind a firewall. The proposed solution does not require modification of the existing security infrastructure. This offers a fast deployment but does not provide maximum security at the boundary.

Architecture

The architecture of the security system is presented in Figure 17. This architecture is very classic, the service is connected to the Engine Inside. The Engine Outside act as a proxy in the DMZ for the Engine Inside. Connections are established to the Engine Outside and then messages are routed to the Engine Inside.

E-speak offers end-to-end security so the number of engines relaying messages is irrelevant. The security (Confidentiality, Integrity and Authentication) is established between the services and the clients.

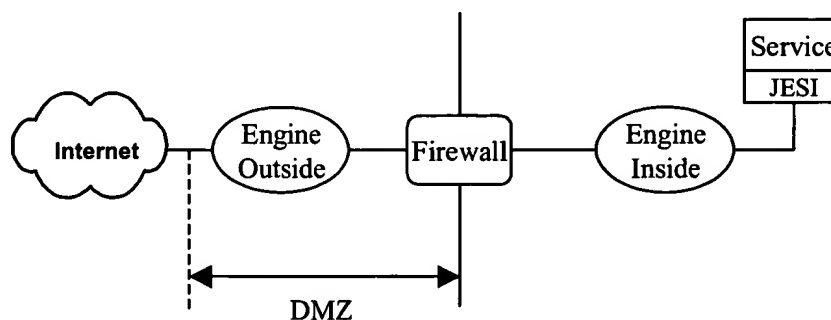


Figure 17 Security Architecture

Firewalls usually offer means to establish outbound connections. HTTP proxies are one of them. They are widely deployed and offer through the HTTP Connect method a way to establish TCP connection to external systems. SOCKS V4 servers offer the same functionality but support stronger authentication mechanism.

The requirements on the service provider side are as follows:

- The requirements on the client side are as follows:

- The HTTP proxy approach is possible only if the proxy allows for the HTTP Connect method on the Engine Outside port.

164

Deployment using HTTP proxies

Figure 18 and Figure 19 presents classic deployment configuration using e-speak.

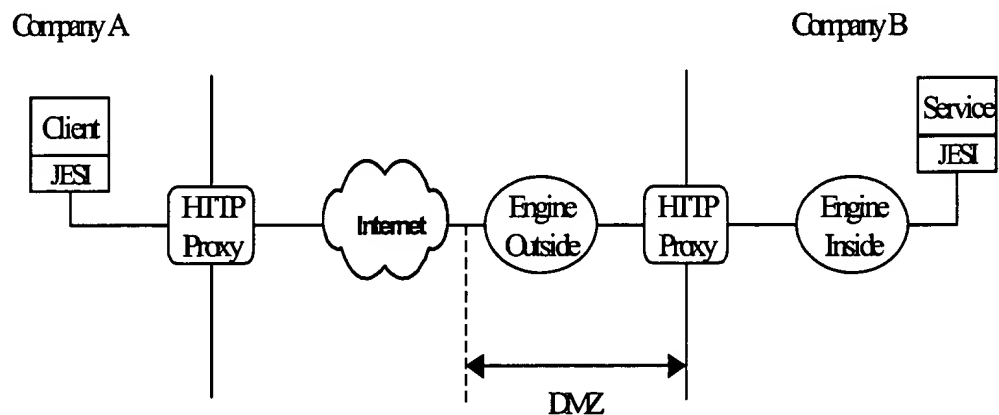


Figure 18 Client connecting directly to the Engine Outside

In this scenario, the client connects directly to the Engine Outside through J-ESI.

Follow these steps:

- 1 Setup the client `espeak.cfg` file with the following properties:
`net.espeak.infra.cci.messaging.webproxyname=<proxy name>`
`net.espeak.infra.cci.messaging.webproxyport=<proxy port>`
 This explicitly tells the client to open a connection through the web proxy.
- 2 Setup the Engine Inside `espeak.cfg` file with the following properties:
`net.espeak.infra.core.connector.webproxyname=<proxy name>`
`net.espeak.infra.core.connector.webproxyport=<proxy port>`
- 3 The service needs to explicitly export itself to the Engine Outside using its `RemoteServiceManager`.

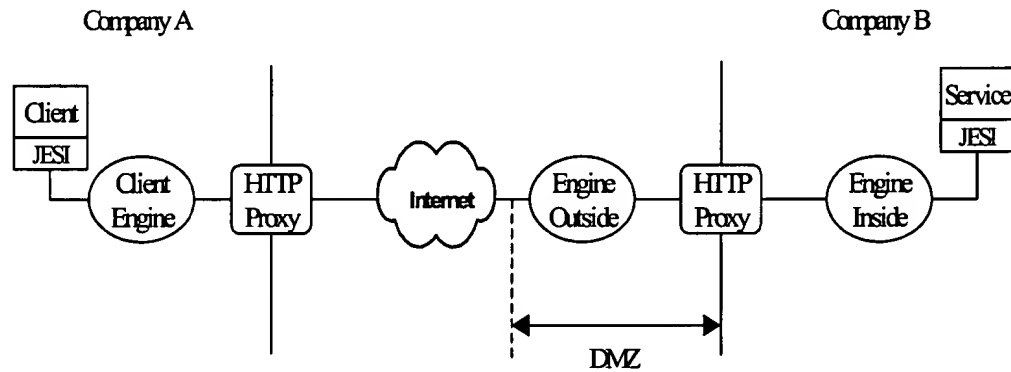


Figure 19 Client connecting to the Engine Outside using a local engine

In this scenario, the client connects to a local engine, which in turn connects to the Engine Outside.

Follow these steps:

- 1 Setup the Client Engine `espeak.cfg` file with the following properties:
`net.espeak.infra.core.connector.webproxyname=<proxy name>`
`net.espeak.infra.core.connector.webproxyport=<proxy port>`
- 2 Setup the Engine Inside `espeak.cfg` file with the following properties:
`net.espeak.infra.core.connector.webproxyname=<proxy name>`
`net.espeak.infra.core.connector.webproxyport=<proxy port>`
- 3 The service explicitly exports itself to the Engine Outside using its `RemoteServiceManager`.
- 4 The client explicitly imports the service from the Engine outside using its `RemoteServiceManager`.

NOTE: As explicit import and export are awkward, another alternative is to run an advertising service in the DMZ attached to the Engine Outside and use it to do the import and export implicitly. With this modification, the above scenario changes as follows:

- 1 setup the Client Engine `espeak.cfg` file as above.

- 2 setup the Engine Inside espeak.cfg file as above.
- 3 The service is advertised in the advertising service in the DMZ. This leads to an implicit export from the Engine Inside to the Engine Outside.
- 4 The client finds the service by looking up in the DMZ advertising service. This leads to an implicit import from the Engine Outside to the Client Engine.

The scenario in Figure 19 can be modified similarly.

Deployment using SOCKS Server

The SOCKS protocol is supported internally by the VMs. No specific configuration is needed except for the VM configuration itself.

Figure 20 shows one scenario where the SOCKS servers are used by the client and the service provider.

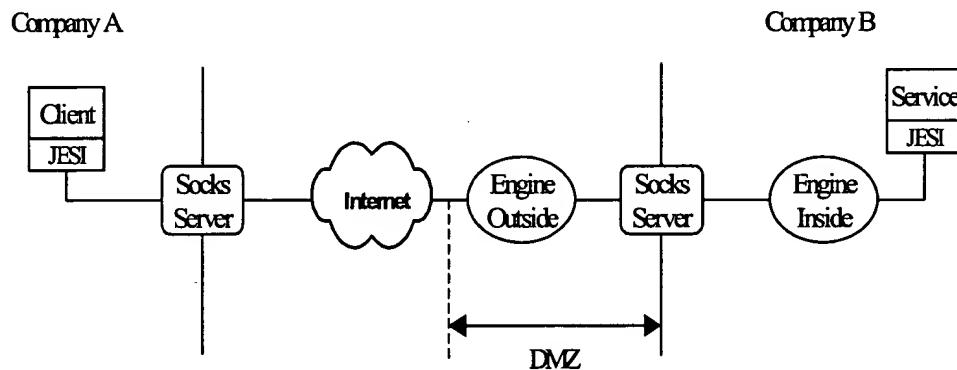


Figure 20 Using SOCKS servers

Connector

In some deployment cases, it is possible that installing the Engine Outside in the DMZ is perceived as too complicated or impossible.

E-speak introduces a new way for service providers to reach service consumers. The connector is the central point of the system. The connector is the Engine Outside moved in a DMZ of a trusted party.

Figure 21 describe the deployment scenario for the connector.

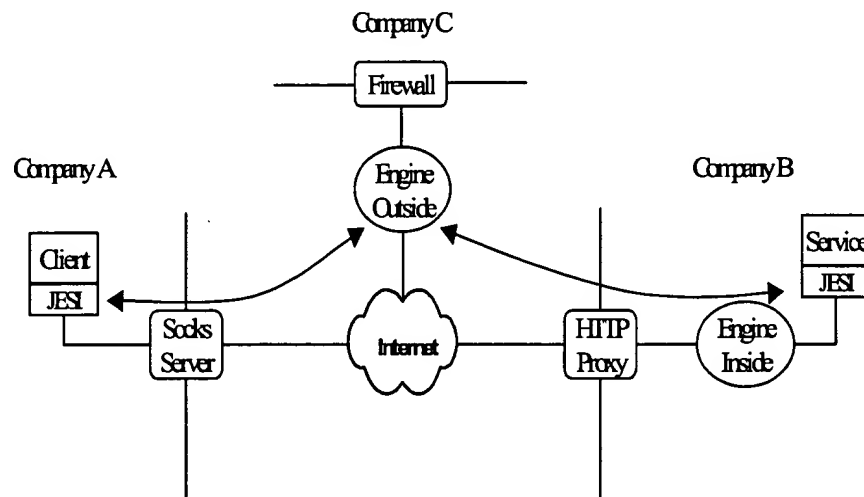


Figure 21 Deployment of a connector in a trusted party's DMZ

Company C is a trusted by Company B to host its services' metadata. When Company B wants to expose its services to the outside, it connects the Engine Inside to the Engine Outside (Connector) and exports the description of the available services.

When the clients wants to access a service, its connect to the connector and search for the given service and interact with it transparently.

Clearly Company C needs to be trusted by the service provider because it exposes the service to the outside world.

An example of the connector is the e-service village where service providers and services consumers can meet and establish relationship.

Appendix H Management

Currently system management in e-speak is undergoing a transition from a traditional network object model to a document exchange model. Infrastructure and support for this model is still being developed and collaboration between parties on common schemas and mechanisms is in the early phases. This document therefore is very much work in progress and will remain as such until a variety of inter-related elements within e-speak become available and reach stability.

Introduction

This appendix describes the infrastructure and general philosophy of how web based management tools and an XML management interaction framework are provided in e-speak.

System Management in e-speak currently consists of four things:

- A managed service model.
- XML Schemas and dialogs that enable management.
- Client support for making services manageable.
- A set of tools that provide access to the management services from the web.

This document describes the first, third and fourth of the above with code examples.

Managed Service Model

The managed service model is simply two concepts that underpin the manageability of e-speak services and e-speak clients:

- Managed State: a defined service state embodying the life cycle of a service.
- Managed Variable Table: sets of values that can be affected by a manager for the purposes of configuration and control.

For this generic model to be useful, a client must map its service specific behavior onto this model and expose this model to a management agent. As we see later, the life cycle in the managed service model has many states, including *initializing*, *ready*, and *running*. It is entirely up to the service writer how these states are related to the service specific behavior.

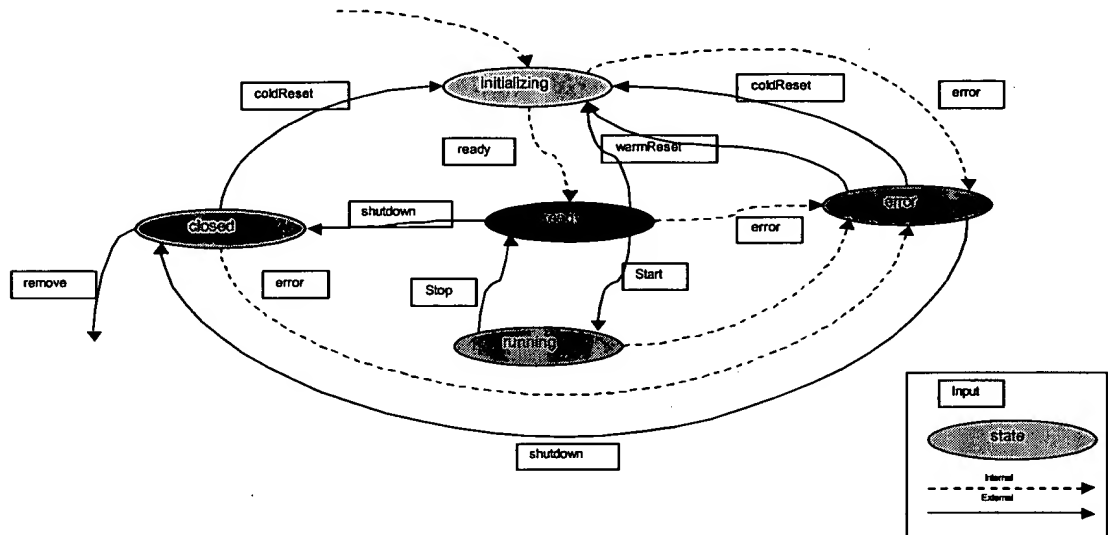
For example a service might have no need of an initialization phase but must instantaneously pass this state to conform to the model. A service may also be in various operational states while running, but *running* is the only way to express its condition from a management point of view.

In summary then, the managed service model embodies a view of service behavior that is potentially common to many services. Therefore some custom management agents must deal with issues of service behavior that fall outside this model.

0973307.120000

Managed Life Cycle

The full state transition diagram is as follows:



State Descriptions

Initializing: The internal dynamic state of the service is being constructed, for example: a policy manager is being queried for configuration information and resources are being discovered via search recipes or yellow pages servers. When the service finishes this work it moves asynchronously into the ready or error states.

Ready: The service is in a ready to run situation, this state is also equivalent to a stopped or paused state.

Running: The service is running and responding to methods invoked on its operational interfaces. If an error occurs that implies that the service cannot continue to run, it should move into the error state.

Error: The service has a problem and is awaiting management action on what to do next.

[illegible]

An input is the trigger that causes a state transition to occur. In any given state, there is a defined set of permissible inputs that are available—only those that are depicted in the diagram as leaving the current state and connecting with the next state. To attempt to perform any other transition is illegal. Note that many inputs can have the same name (e.g. error) but there is no ambiguity as long as the originating state is different.

The available inputs are as follows:

stop: move into the ready state. Stop handling invocations on operational interfaces.

error: move into the error state, this transition is valid from any state.

coldReset: cause a from complete reinitialization of the service and move into the initializing state. The only exemption is that resources that are already registered should not be reregistered.

remove: cause the service to remove itself from existence. Any non-persistent resources should be deregistered from the repository.

Managed Variable Tables

A managed variable table is at its simplest a table of name/string value pairs that exist within the client but to which a manager has some level of access. Thus, a management agent can control those aspects of a services behavior that is affected by those variables to which it has access.

There is a degree of configurability associated with managed variables and their variables that permit something more sophisticated than the simple get and set operations one would expect to find.

Each table itself has a name to distinguish it from other tables. As we shall see later, the managed service model itself provides for two such tables.

The most simple usage case for a managed variable would be for a variable which the management agent has only read access but which the client varies as necessary. The manager can monitor the changes (see events later) such that an operator who understands the meaning of this variable (e.g. `secondsToDetonation`) might gain some information.

The next level of sophistication is to enable the operator to affect the services behavior by modifying the value of the variable. However, some service configuration is only of practical use during initialization (e.g. some resource allocation parameter). Knowing the instantaneous value of such a variable does not explain the clients behavior (if it had been changed) and modifying it has no effect.

To account for this, a managed variable can have one or more values. The compulsory value is its "live" value i.e. the value that is contributing to its current behavior. In the simplest case this is the only value that a variable has.

However, a variable can have other values, which are in effect "scheduled" values and associated with the various reset operations in the client's life cycle.

For example, consider the following managed variable found in some unreasonably hazardous device:

Variable Name	Values		
	Value Condition	Remote Access	Value
secondsToDetonation	Live Value	Read Only	42
	On Cold Reset	Read Only	60
	On Warm Reset	Read/Write	60

Let's assume it takes some operator three minutes to reach safety or the devices off switch. The operator brings up his management console and notices that the current value of secondsToDetonation is 42, and dropping. Clearly the circumstances are undesirable: the current value does not provide long enough to escape without injury. Increasing this value to three minutes or more and running away is ideal but the value is read only.

Glancing down the list the operator notices that there are two reset values and performs a cold reset. After the device has re-booted, and re-loaded the counters default initial value (perhaps in ROM) it promptly starts counting down from 60 seconds. Still not good but at least the operator can try and stay up all night performing cold resets every 59 seconds or less.

Having bought some time the operator looks a little more closely at the variable. The warm reset value is writable. Now the operator can set this to thirty minutes and go and disconnect the power supply.

Clearly a contrived example but it does demonstrate how relatively complex interactions between initialization, configuration value and behavior can be represented.

A more likely scenario is where a client has a set of variables that configure its behavior that must all be changed synchronously. A management agent can modify its warm reset values at its leisure (being denied access to the live values) and then perform a warm reset.

There is a restriction on variable table usage:

Uniqueness: names in a variable table must be unique within that table. It is not possible to implement lists by having many entries with the same name.

Configuration Parameter Table

The configuration parameter table is an instance of a managed variable table with a reserved name that identifies it as such. The table holds generic configuration data for the client.

Resource Table

The resource table is another instance of a managed variable table, identical in behavior to the configuration parameter table except that the names in the client's table refer to other services with which the client has some relationship. For example, if a particular client makes use of a mail service then this relationship can be made visible to a management agent through the resource table. Thus a management agent might reconfigure the client to use an alternative but equivalent service. While there might seem no obvious need to separate out this particular aspect of configuration, doing so makes it possible for a management agent to discover the topology and integrity of a network of connected services without the need for service specific interpretation of the variable table (all entries in the resource table are resources).

The name used for an entry in a resource table can be any symbolic name the client chooses, while the value must be the valid e-speak URL of the actual service.

System Management Events

Events can be considered as notifications of some significant occurrence. In the context of system management, events must exist to convey notification of some change of state with regards the Managed Service Model. The following events are therefore defined:

ManagedServiceStateChange: This event conveys information regarding some managed service state transition to any interested management agent. In this case, the event is the name of the input provided to the management life cycle FSM. Clearly, the management agent must have known what the previous state was in order that this information be meaningful.

ManagedVariableValueChange: This event conveys information regarding a value modification in a clients configuration or resource table.

Managed Service Programmers Guide

Writing a Simple Managed Service

In this section we create a simple managed service¹.

First, to get familiar with the tools we look at the most minimal Java program that is "manageable" through e-speak.

A Minimal Service

Consider the following program:

```
import
net.espeak.management.managedservice.simplemanagedservice.SimpleXMLManaged
Service;
import net.espeak.infra.cci.exception.ESEException;
import net.espeak.jesi.management.ServiceContext;

public class VerySimpleExample extends SimpleXMLManagedService {
    public static final String SERVICE_NAME = "VerySimpleService";
    public static final String LOCALHOST = "127.0.0.1";
    public static final int COREPORT = 12346;

    public VerySimpleExample() throws ESEException {
        super(new ServiceContext(LOCALHOST, COREPORT, "tcp"), SERVICE_NAME);
        makeManageable();
    }

    protected void stateChangeOccurred(String transition, String oldState,
        String newState) {}

    protected void resourceChangeOccurred(String resource, String oldValue,
        String newValue) {}
}
```

- ¹ For reference, a fully working example of a managed service is supplied in:
net.espeak.management.managedservice.simplemanagedservice.ExampleService

```

        protected void variableChangeOccurred(String variable, String oldValue,
            String newValue) {}

        protected boolean acceptStateChange(String transition, String oldState,
            String newState) {
            return true;
        }

        protected boolean acceptVariableChange(String variable, String oldValue,
            String newValue) {
            return true;
        }

        protected boolean acceptResourceChange(String resource, String oldValue,
            String newValue) {
            return true;
        }

        public static void main(String[] args) {
            try {
                new VerySimpleExample();
            } catch (ESEException e) {
                e.printStackTrace();
            }
        }
    }

```

VerySimpleExample (above) subclasses SimpleXMLManagedService, a utility class that in most circumstances provides adequate manageability support. We consider more sophisticated examples later.

For now let's just consider the constructor:

```

    public VerySimpleExample() throws ESEException {
        super(new ServiceContext(LOCALHOST, COREPORT, "tcp"),
            SERVICE_NAME);
        makeManageable ();
    }

```

First, the super class is initialized with a connection to the local core², second the **makeManageable()** method makes the service visible to any management agents.

Managing the Service

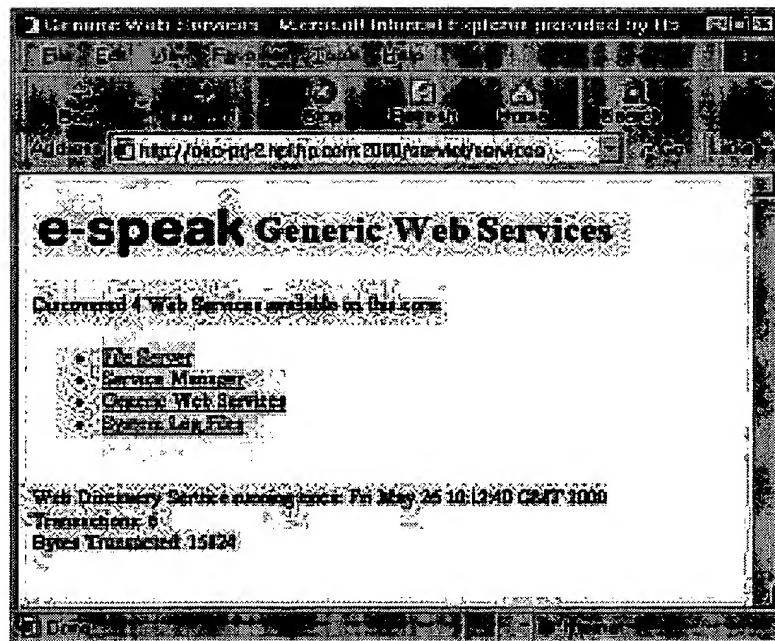
We are now ready to try the program out, but first we must have a core running. For convenience, there is a configuration file which starts all the services required for management. Go to your e-speak installation directory and type:

```
espeak -i config\management\xmlmanagement.ini
```

After a few moments, run a web browser and go to:

<http://127.0.0.1:2000/servlet/services>

You see a screen similar to the following:



- 2 For simplicity, the host and port are hardwired into the example and assume the core is running on the local host and uses port 12346, which is the value in the default version of xmlmanagement.ini. These values can be changed but the host must be the host name/IP address of a host running a core and the port number must match that used by the core.

This is a list of the visible “web enabled” services on your local core, in this case the services started by xmlmanagement.ini.

Click the Service Manager link and you see the following page:



This is the main service manager screen and displays a list of all the manageable services to be found on your local core.

Note that while we call this the Service Manager, it should be remembered that it is clearly a web based interface onto the actual Service Manager. While the service manager is aware of the status of services in real time, the browser is not; so you should use the “Update” buttons provided to keep the display current. For related reasons, navigate between Service Manager screens using the links provided and avoid the use of the back button, particularly if you actively interacting with the services through the service manager.

Now we are ready to run **VerySimpleExample**. Compile the program and run it. After a few moments go back to the service manager screen and click the Update button. Now you see your program available for management:



Click the Management link next to **VerySimpleService**:

000021" 420E260



This is the management page for your service. At the moment, your service is not being managed so click the Manage Service button:



So what does this mean? Earlier in the document we discussed the concept of the Managed Life Cycle of a service. If you look back at the state diagram, you see that the start state was *initializing*. Because we have not changed service state in our program, we remain in this state forever.

Changing Service State

Now let's try something slightly more interesting. Consider this next program, very similar to the last:

```
import
net.espeak.management.managedservice.simplemanagedservice.SimpleXMLManaged
Service;
import net.espeak.infra.cci.exception.ESEException;
import net.espeak.jesi.management.ServiceContext;
import
net.espeak.management.managedservice.managedstate.ManagedServiceStateMachi
ne;
```



```
public class VerySimpleExample2 extends SimpleXMLManagedService {
    public static final String SERVICE_NAME = "VerySimpleService2";
    public static final String LOCALHOST = "127.0.0.1";
    public static final int COREPORT = 12346;

    public VerySimpleExample2() throws ESEException {
        super(new ServiceContext(LOCALHOST, COREPORT, "tcp"), SERVICE_NAME);
        makeManageable();
        performTransition(ManagedServiceStateMachine.TO_READY_TRANSITION);
    }

    protected void stateChangeOccurred(String transition, String oldState,
        String newState) {}

    protected void resourceChangeOccurred(String resource, String oldValue,
        String newValue) {}

    protected void variableChangeOccurred(String variable, String oldValue,
        String newValue) {}

    protected boolean acceptStateChange(String transition, String oldState,
        String newState) {
        return true;
    }

    protected boolean acceptVariableChange(String variable, String oldValue,
        String newValue) {
        return true;
    }

    protected boolean acceptResourceChange(String resource, String oldValue,
        String newValue) {
        return true;
    }

    public static void main(String[] args) {
        try {
            new VerySimpleExample2();
        } catch (ESEException e) {
            e.printStackTrace();
        }
    }
}
```

Here, the only real difference is in the constructor, where we have added the line:

```
performTransition(ManagedServiceStateMachine.TO_READY_TRANSITION);
```

Have another look at the service state diagram and you see an internal transition taking the service to the *Ready* state. The line of code, above, causes this transition to occur. Lets compile and run this program to see what happens when viewed through the Service Manager.

After the VerySimpleService2 is running, go to back to the Service Manager and click the *All Services* link and click the *Update* button. Your new program is available for management:



As before, go to the Services Management screen and click the *Manage Service* button. You see the following:



The state of your service is now listed as ready and three new buttons have appeared. Looking back at the service state diagram, you can see that when a service is in the *Ready* state a manager can request that it can start, stop or perform a warm reset. Click the Start and the following screen appears:



Clicking the Stop button takes you back to the previous screen. All very interesting, but how does this relate to the program we're running? At the moment the program isn't affected at all by these management changes.

Adding Custom Service Behavior

So far the service doesn't do anything interesting, so let's take a look at how we can improve the situation:

```
import
net.espeak.management.managedservice.simplemanagedservice.SimpleXMLManaged
Service;
import net.espeak.infra.cci.exception.ESEException;
import net.espeak.jesi.management.ServiceContext;
import
net.espeak.management.managedservice.managedstate.ManagedServiceStateMachi
ne;
```

```

public class VerySimpleExample3 extends SimpleXMLManagedService
    implements Runnable {
    public static final String SERVICE_NAME = "VerySimpleService3";
    public static final String LOCALHOST = "127.0.0.1";
    public static final int COREPORT = 12346;
    protected boolean running = false;
    protected Thread runThread = null;

    public VerySimpleExample3() throws ESEException {
        super(new ServiceContext(LOCALHOST, COREPORT, "tcp"), SERVICE_NAME);
        makeManageable();
        performTransition(ManagedServiceStateMachine.TO_READY_TRANSITION);
    }

    public void run() {
        while (true) {
            synchronized (runThread) {
                if (!running) {
                    return;
                }
            }
            System.out.println("Running " + System.currentTimeMillis());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {}
        }
    }

    protected synchronized void stateChangeOccurred(String transition,
        String oldState, String newState) {
        if (newState.equals(ManagedServiceStateMachine.RUNNING_STATE)) {
            running = true;
            runThread = new Thread(this);
            runThread.start();
        } else if (runThread != null) {
            synchronized (runThread) {
                running = false;
            }
            try {
                runThread.join();
            } catch (InterruptedException e) {}
            runThread = null;
        }
    }

    protected void resourceChangeOccurred(String resource, String oldValue,

```

```

        String newValue) {}

    protected void variableChangeOccurred(String variable, String oldValue,
        String newValue) {}

    protected boolean acceptStateChange(String transition, String oldState,
        String newState) {
        return true;
    }

    protected boolean acceptVariableChange(String variable, String oldValue,
        String newValue) {
        return true;
    }

    protected boolean acceptResourceChange(String resource, String oldValue,
        String newValue) {
        return true;
    }

    public static void main(String[] args) {
        try {
            new VerySimpleExample3();
        } catch (ESEException e) {
            e.printStackTrace();
        }
    }
}

```

Here we have given the program a runnable thread and added some code to the **stateChangeOccurred()** method. This method is called after any change of service state. If you compile and run this program, then when you start and stop it from the Service Manager you also see the thread started and stopped.

Managing Variables

It is also possible to manage the data within your service. Consider the next derivative of the service:

```

import
net.espeak.management.managedservice.simplemanagedservice.SimpleXMLManaged
Service;
import net.espeak.infra.cci.exception.ESEException;
import net.espeak.jesi.management.ServiceContext;

```

```

import
net.espeak.management.managedservice.managedstate.ManagedServiceStateMachi
ne;

public class VerySimpleExample4 extends SimpleXMLManagedService
    implements Runnable {
    public static final String SERVICE_NAME = "VerySimpleService4";
    public static final String LOCALHOST = "127.0.0.1";
    public static final int COREPORT = 12346;
    public static final String MESSAGE_VAR = "Message";
    protected boolean running = false;
    protected Thread runThread = null;
    protected String message = "running";

    public VerySimpleExample4() throws ESEException {
        super(new ServiceContext(LOCALHOST, COREPORT, "tcp"), SERVICE_NAME);
        createVariable(MESSAGE_VAR, message, true);
        makeManageable();
        performTransition(ManagedServiceStateMachine.TO_READY_TRANSITION);
    }

    public void run() {
        while (true) {
            synchronized (this) {
                if (!running) {
                    return;
                }
                System.out.println (message + " " +
System.currentTimeMillis());
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {}
        }
    }

    protected synchronized void stateChangeOccurred(String transition,
        String oldState, String newState) {
        if (newState.equals(ManagedServiceStateMachine.RUNNING_STATE)) {
            running = true;
            runThread = new Thread(this);
            runThread.start();
        } else if (runThread != null) {
            running = false;
            try {
                runThread.join();
            }
        }
    }
}

```

```

        } catch (InterruptedException e) {}
        runThread = null;
    }
}

protected void resourceChangeOccurred(String resource, String oldValue,
String newValue) {}

protected synchronized void variableChangeOccurred(String variable,
String oldValue,
String newValue) {
    if (variable.equals(MESSAGE_VAR))
    {
        message = newValue;
    }
}

protected boolean acceptStateChange(String transition, String oldState,
String newState) {
    return true;
}

protected boolean acceptVariableChange(String variable, String oldValue,
String newValue) {
    return true;
}

protected boolean acceptResourceChange(String resource, String oldValue,
String newValue) {
    return true;
}

public static void main(String[] args) {
    try {
        new VerySimpleExample4();
    } catch (ESEException e) {
        e.printStackTrace();
    }
}
}

```

Here we have added a normal class variable called "message" which we print in the run method. What we have also done in the constructor is create a *Managed Variable* that informs the system manager that such a variable exists, i.e. what it's name is, what it's initial value is and whether it can be remotely modified:

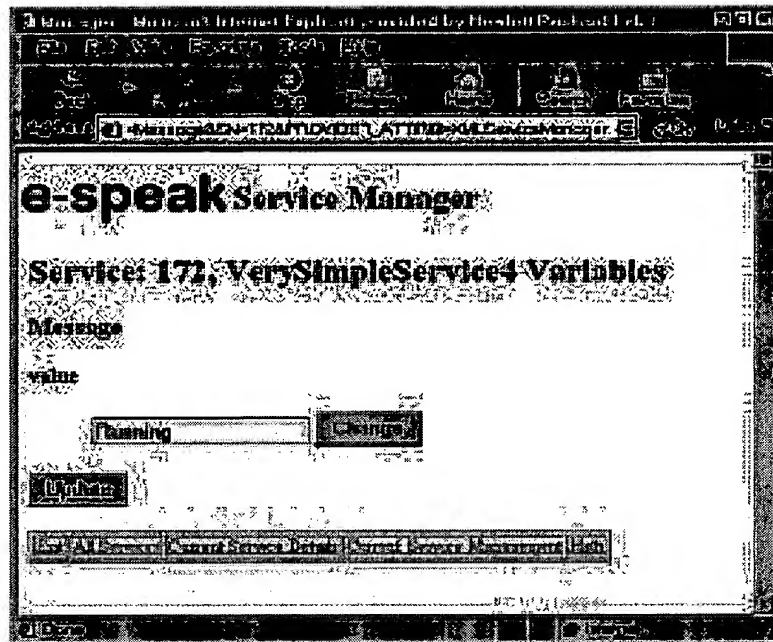

```
createVariable(MESSAGE_VAR, message, true);
```

This is sufficient to make the variable visible to the Service Manager, but we also want our program to reflect changes to the value of this variable made by the service manager. To achieve this we have added some code to the **variableChangeOccurred()** method. This code simply verifies which *Managed Variable* has changed and ensures that the appropriate action is taken. In this case, it simply updates the local message variable.

Viewing this service through the Service Manager looks as follows:



You can see that in addition to the service state, our managed variable is visible. Clicking the variable name opens the following screen:



Here you can change the value of the variable. Try changing the value and observe the output of your service change while it is running.

Dynamic Variables

So now we can change the value of our variables remotely, but what would happen if we changed the value locally? If the management system is to know the state of your service from second to second then obviously it needs to be told. This next example uses an additional managed variable, which is updated locally:

```
import
net.espeak.management.managedservice.simplemanagedservice.SimpleXMLManaged
Service;
import net.espeak.infra.cci.exception.ESEException;
import net.espeak.jesi.management.ServiceContext;
import
net.espeak.management.managedservice.managedstate.ManagedServiceStateMachi
ne;
```

```

import java.util.Date;

public class VerySimpleExample5 extends SimpleXMLManagedService
    implements Runnable {
    public static final String SERVICE_NAME = "VerySimpleService5";
    public static final String LOCALHOST = "127.0.0.1";
    public static final int COREPORT = 12346;
    public static final String MESSAGE_VAR = "Message";
    public static final String TIME_VAR = "Time";
    protected boolean running = false;
    protected Thread runThread = null;
    protected String message = "running";

    public VerySimpleExample5() throws ESEException {
        super(new ServiceContext(LOCALHOST, COREPORT, "tcp"), SERVICE_NAME);
        createVariable(MESSAGE_VAR, message, true);
        createVariable(TIME_VAR, new Date ().toString (), false);
        makeManageable();
        performTransition(ManagedServiceStateMachine.TO_READY_TRANSITION);
    }

    public void run() {
        while (true) {
            synchronized (this) {
                if (!running) {
                    return;
                }
                System.out.println (message + " " +
System.currentTimeMillis());
                super.setVariable(TIME_VAR, new Date ().toString ());
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {}
        }
    }

    protected synchronized void stateChangeOccurred(String transition,
        String oldState, String newState) {
        if (newState.equals(ManagedServiceStateMachine.RUNNING_STATE)) {
            running = true;
            runThread = new Thread(this);
            runThread.start();
        } else if (runThread != null) {
            running = false;
            try {

```

```

        runThread.join();
    } catch (InterruptedException e) {}
    runThread = null;
}

protected void resourceChangeOccurred(String resource, String oldValue,
String newValue) {}

protected synchronized void variableChangeOccurred(String variable,
String oldValue,
String newValue) {
    if (variable.equals(MESSAGE_VAR))
    {
        message = newValue;
    }
}

protected boolean acceptStateChange(String transition, String oldState,
String newState) {
    return true;
}

protected boolean acceptVariableChange(String variable, String oldValue,
String newValue) {
    return true;
}

protected boolean acceptResourceChange(String resource, String oldValue,
String newValue) {
    return true;
}

public static void main(String[] args) {
    try {
        new VerySimpleExample5();
    } catch (ESEException e) {
        e.printStackTrace();
    }
}
}

```

Here we have added a new *Managed Variable* that represents the time, which we update in the run method of our service. Viewing our service through the Service Manager now looks something like this:



Here we can see our new (read only) Time variable. Now when you start the service, each time you click the Update button, you see the most recent value of the time variable.

06-07

Appendix I Exceptions

There are three classes of checked exceptions that can be thrown by the J-ESI APIs. They are:

- `ESInvocationException`
- `ESServiceException`
- `ESLibException`

ESInvocationException

`ESInvocationException` is thrown usually when the engine (or the core) detects a problem with the request. Examples of such exceptions are quota exhausted or stale entry.

- `QuotaExhaustedException`, as the name implies, is thrown when there is no more space in the allocated quota of the client to hold any entries.
- `StaleEntryAccessException` is thrown when the service which the request is destined for is stale and the request cannot be delivered to the service.

For a detailed listing of all `ESInvocationExceptions`, see “Exceptions” in Chapter 8 of the e-speak Architectural Specifications, version 3.01.

ESServiceException

ESServiceException happens when the core has delivered a request to the service, but the service has detected an error with the request. Some of the service exceptions thrown by core services are, `LookupFailedException` thrown by the finder service when any lookup fails, `InvalidNameException` thrown by the folder service when any of the names passed is invalid.

Any user specific exception defined also should extend `ESServiceException`. For example, if some one writes a printer service which can throw a `OutOfPaperException`, this should extend `ESServiceException`.

For a detailed listing of all these exceptions, see "Exceptions" in Chapter 8 of the e-speak Architectural Specifications, version 3.01.

ESLibException

ESLibException is usually thrown by the library itself when it has detected an error condition. For example, `CoreNotFoundException` is thrown when no core exists in the `hostname:portnumber` specified.

Apart from these exceptions, `ESRuntimeException` is thrown to indicate a runtime failure. `ESLibRuntimeException` is a subclass of `ESRuntimeException` and is thrown in unexpected behavior conditions.

5

10

Appendix C

Description of Files on CD-ROM in CD-ROM Appendix

09733027.120800

test

Volume in drive E is 001206_1632
Volume Serial Number is 33CE-90D1

Directory of E:\

12/06/00	04:32p	<DIR>	.
12/06/00	04:32p	<DIR>	..
12/06/00	04:33p	<DIR>	e-speak-src_991217
3 File(s)			0 bytes

Directory of E:\e-speak-src_991217

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	platform
3 File(s)			0 bytes

Directory of E:\e-speak-src_991217\platform

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	ES
3 File(s)			0 bytes

Directory of E:\e-speak-src_991217\platform\ES

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	bin
12/06/00	04:33p	<DIR>	config
12/15/99	04:00p		54,294 configure
12/15/99	04:00p		10,314 configure.in
12/07/99	04:25p		53 configure.nt
12/06/00	04:33p	<DIR>	contrib
12/07/99	04:25p		15,413 COPYING
12/07/99	04:25p		23,237 COPYING.LIB
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	extern
12/07/99	04:25p		20,163 HowTo.html
12/15/99	04:00p		4,224 include.mk.in
12/15/99	04:00p		13,149 Makefile.in
12/07/99	04:25p		1,509 make.rules
12/06/00	04:33p	<DIR>	samples
12/06/00	04:33p	<DIR>	src
12/06/00	04:33p	<DIR>	tutorial
19 File(s)			142,356 bytes

test

Directory of E:\e-speak-src_991217\platform\ES\bin

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		2,040 envmake
12/07/99	04:25p		2,007 envmake.bat
12/07/99	04:25p		2,673 envset
12/07/99	04:25p		27 eshome.bat
12/07/99	04:25p		7,292 genenv.pl
12/15/99	04:00p		17,909 install.pl
12/07/99	04:25p		35,269 run.pl
10 File(s)			67,217 bytes

Directory of E:\e-speak-src_991217\platform\ES\bin\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		321 Entries
12/16/99	06:17p		31 Repository
12/16/99	06:17p		46 Root
12/16/99	06:17p		846 Template
6 File(s)			1,244 bytes

Directory of E:\e-speak-src_991217\platform\ES\config

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	co
12/07/99	04:25p		22 col
12/07/99	04:25p		22 co2
12/07/99	04:25p		22 codef
12/07/99	04:25p		133 core.ini
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		1,039 default.ini
12/07/99	04:25p		149 espeak_ldap.prop
12/06/00	04:33p	<DIR>	management
12/07/99	04:25p		25 multiplecf.cfg
12/07/99	04:25p		510 repository.ini
12/07/99	04:25p		17 sample.cfg
14 File(s)			1,939 bytes

Directory of E:\e-speak-src_991217\platform\ES\config\co

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		22 col
12/07/99	04:25p		22 col0

```

12/07/99 04:25p      test
12/07/99 04:25p      22 col.stress
12/07/99 04:25p      22 co2
12/07/99 04:25p      22 co2.stress
12/07/99 04:25p      22 co3
12/07/99 04:25p      22 co3.stress
12/07/99 04:25p      22 co4
12/07/99 04:25p      30 co.SVEN.OMNIBOOK1
12/07/99 04:25p      22 co4.stress
12/07/99 04:25p      22 co5
12/07/99 04:25p      30 co.SVEN.OMNIBOOK
12/07/99 04:25p      22 co5.stress
12/07/99 04:25p      22 co6
12/07/99 04:25p      24 co.PHIL.OMNI3
12/07/99 04:25p      22 co7
12/07/99 04:25p      22 co8
12/07/99 04:25p      22 co9
12/07/99 04:25p      24 co.PHIL.OMNI4
12/07/99 04:25p      32 co.CHIA.CHOWCHOW1
12/07/99 04:25p      24 co.PHIL.IU1
12/07/99 04:25p      24 co.PHIL.IU24
12/07/99 04:25p      33 co.SVEN.KAYAK1
12/07/99 04:25p      22 co.CHIA.localhost
12/07/99 04:25p      22 co.MYCO3
12/07/99 04:25p      24 co.PHIL.OMNI2
12/07/99 04:25p      32 co.CHIA.CHOWCHOW
12/07/99 04:25p      24 co.PHIL.IU23
12/07/99 04:25p      33 co.SVEN.KAYAK
12/07/99 04:25p      22 co.MYCO2
12/07/99 04:25p      24 co.PHIL.OMNI
12/07/99 04:25p      24 co.PHIL.IU22
12/07/99 04:25p      22 co.MYCO1
12/07/99 04:25p      30 co.CHIA.OMNIBOOK1
12/07/99 04:25p      24 co.PHIL.IU2
12/07/99 04:25p      22 co.MYCO
12/07/99 04:25p      30 co.CHIA.OMNIBOOK
12/06/00 04:33p      <DIR>      CVS
                                40 File(s)      906 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\config\co\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:18p      1,736 Entries
12/16/99 06:18p      37 Repository
12/16/99 06:18p      46 Root
12/16/99 06:18p      846 Template
                                6 File(s)      2,665 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\config\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                      420 Entries
12/16/99  06:18p                      32 Entries.Log
12/16/99  06:18p                      34 Repository
12/16/99  06:18p                      46 Root
12/16/99  06:18p                      846 Template
                                7 File(s)          1,378 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\config\management

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p                      22 co2.dat
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      html
12/07/99  04:25p                      980 management.ini
                                6 File(s)          1,002 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\config\management\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                      98 Entries
12/16/99  06:18p                      14 Entries.Log
12/16/99  06:18p                      45 Repository
12/16/99  06:18p                      46 Root
12/16/99  06:18p                      846 Template
                                7 File(s)          1,049 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\config\management\html

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      images
12/07/99  04:25p                      687 main.html
12/07/99  04:25p                      882 servlet.properties
                                6 File(s)          1,569 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\config\management\html\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                      104 Entries
```

test

12/16/99	06:18p	16	Entries.Log
12/16/99	06:18p	50	Repository
12/16/99	06:18p	46	Root
12/16/99	06:18p	846	Template
	7 File(s)	1,062	bytes

Directory of E:\e-speak-src_991217\platform\ES\config\management\html
\images

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		6,840 Closed.gif
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		6,910 Error.gif
12/07/99	04:25p		6,852 Initializing.gif
12/07/99	04:25p		5,058 logo.gif
12/07/99	04:25p		6,897 Ready.gif
12/07/99	04:25p		6,880 Running.gif
	9 File(s)		39,437 bytes

Directory of E:\e-speak-src_991217\platform\ES\config\management\html
\images\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		288 Entries
12/16/99	06:18p		57 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
	6 File(s)		1,237 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	browser
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	gsysmon
12/15/99	04:00p		245 Makefile
12/06/00	04:33p	<DIR>	sysmon
12/06/00	04:33p	<DIR>	vfs
	8 File(s)		245 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		3,280 Browser.java

```

                                test
12/06/00  04:33p      <DIR>      core
12/06/00  04:33p      <DIR>      CVS
12/15/99   04:00p                      991 Makefile
12/06/00  04:33p      <DIR>      tree
12/06/00  04:33p      <DIR>      ui
12/06/00  04:33p      <DIR>      util
                                9 File(s)          4,271 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\core

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99   04:40p                      12,861 CoreBrowser.java
12/06/00  04:33p      <DIR>      CVS
12/07/99   04:40p                      390 Makefile
                                5 File(s)          13,251 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\core\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99   06:18p                      101 Entries
12/16/99   06:18p                      48 Repository
12/16/99   06:18p                      46 Root
12/16/99   06:18p                      846 Template
                                6 File(s)          1,041 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99   06:18p                      97 Entries
12/16/99   06:18p                      54 Entries.Log
12/16/99   06:18p                      43 Repository
12/16/99   06:18p                      46 Root
12/16/99   06:18p                      846 Template
                                7 File(s)          1,086 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\tree

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99   04:25p                      246 a.gif
12/07/99   04:25p                      247 alert.red.gif
12/07/99   04:25p                      242 alert.black.gif
12/07/99   04:25p                      2,326 apache_pb.gif
12/07/99   04:25p                      1,573 AttributePropertyNode.java

```


12/16/99	06:18p	test
12/16/99	06:18p	4,506 Entries
12/16/99	06:18p	48 Repository
12/16/99	06:18p	46 Root
12/16/99	06:18p	846 Template
	6 File(s)	5,446 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\ui

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		1,816 BrowserApp.java
12/07/99	04:25p		4,628 BrowserFrame.java
12/07/99	04:25p		1,780 BrowserView.java
12/07/99	04:25p		2,397 BrowserToolBar.java
12/07/99	04:25p		4,539 BrowserStatusBar.java
12/07/99	04:25p		1,555 BrowserMenuBar.java
12/07/99	04:25p		5,285 ConnectDlg.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		2,699 FileMenu.java
12/07/99	04:25p		2,842 HelpMenu.java
12/07/99	04:25p		172 left.gif
12/07/99	04:25p		2,081 ListPane.java
12/15/99	04:00p		861 Makefile
12/07/99	04:25p		3,291 NumberField.java
12/07/99	04:25p		172 right.gif
12/07/99	04:25p		2,326 TreePane.java
12/07/99	04:25p		1,310 Utility.java
12/07/99	04:25p		3,722 ViewMenu.java
12/07/99	04:25p		1,388 ViewTypes.java
	21 File(s)		42,864 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\ui\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		923 Entries
12/16/99	06:18p		46 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
	6 File(s)		1,861 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\util

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		2,687 Application.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		8,388 DefaultResourceVisitor.java

```

12/07/99 04:25p      test
12/07/99 04:25p      5,278 ExternalResource.java
12/07/99 04:25p      2,667 FrameworkVocabulary.java
12/07/99 04:25p      1,517 FrameworkException.java
12/07/99 04:25p      3,131 FrameworkContext.java
12/15/99 04:00p      878 Makefile
12/07/99 04:25p      11,438 PrintResourceVisitor.java
12/07/99 04:25p      2,484 PrintTreeFactory.java
12/07/99 04:25p      2,981 ResourceVisitor.java
12/07/99 04:25p      5,682 ResourceProvider.java
12/07/99 04:25p      5,381 ResourceDiscover.java
12/07/99 04:25p      3,180 VocabularyFinder.java
      16 File(s)      55,692 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\browser\util\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:18p      753 Entries
12/16/99 06:18p      48 Repository
12/16/99 06:18p      46 Root
12/16/99 06:18p      846 Template
      6 File(s)      1,693 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:18p      48 Entries
12/16/99 06:18p      63 Entries.Log
12/16/99 06:18p      35 Repository
12/16/99 06:18p      46 Root
12/16/99 06:18p      846 Template
      7 File(s)      1,038 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\gsysmon

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:25p      5,116 BarSurface.java
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:25p      1,806 GraphIntf.java
12/07/99 04:25p      7,034 GraphSurface.java
12/15/99 04:00p      533 Makefile
12/07/99 04:25p      8,439 SysMonGui.java
12/07/99 04:25p      7,198 TGFrame.java
      9 File(s)      30,126 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\contrib\gsysmon\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		305 Entries
12/16/99	06:18p		43 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
		6 File(s)	1,240 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\sysmon

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		386 Makefile
12/07/99	04:25p		8,877 SysMon.java
		5 File(s)	9,263 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\sysmon\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		96 Entries
12/16/99	06:18p		42 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
		6 File(s)	1,030 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	config
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	src
		5 File(s)	0 bytes

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\config

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		20 col
12/07/99	04:25p		20 co2
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		2,895 Makefile
12/07/99	04:25p		1,147 Makefile.bat
12/07/99	04:25p		135 VFSBrowser.ini

```

                                test
12/07/99  04:25p                272 VFSBrowser.prop
12/07/99  04:25p                147 VFSFileStore.ini
12/07/99  04:25p                448 VFSFileStore.prop
12/07/99  04:25p                137 VFSShell.ini
12/07/99  04:25p                272 VFSShell.prop
                                13 File(s)
                                5,493 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\config\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:18p                486 Entries
12/16/99  06:18p                46 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                6 File(s)
                                1,424 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:18p                3 Entries
12/16/99  06:18p                29 Entries.Log
12/16/99  06:18p                39 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                7 File(s)
                                963 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/06/00  04:33p                <DIR>                CVS
12/06/00  04:33p                <DIR>                vfs
                                4 File(s)
                                0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:18p                3 Entries
12/16/99  06:18p                13 Entries.Log
12/16/99  06:18p                43 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                7 File(s)
                                951 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs

test

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> browser
12/06/00 04:33p <DIR> clientapi
12/06/00 04:33p <DIR> CVS
12/06/00 04:33p <DIR> intf
12/07/99 04:25p 450 Makefile
12/06/00 04:33p <DIR> server
12/06/00 04:33p <DIR> shell
12/06/00 04:33p <DIR> util
          10 File(s)          450 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\browser

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:25p 4,924 AttributeViewListener.java
12/07/99 04:25p 31,720 AttributeView.java
12/07/99 04:25p 2,616 AttributeItem.java
12/07/99 04:25p 25,061 Browser.java
12/07/99 04:25p 472 Browser.resources
12/07/99 04:25p 106,734 BrowsePane.java
12/07/99 04:25p 4,296 BrowsePane.resources
12/07/99 04:25p 7,855 BrowseDialog.java
12/07/99 04:25p 11,116 CabinetAttributesDialog.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:25p 2,476 DirectoryNode.java
12/07/99 04:25p 1,614 DragData.java
12/07/99 04:25p 6,742 FileInfo.java
12/07/99 04:25p 9,936 FileStoreListDialog.java
12/07/99 04:25p 7,950 FolderWatcher.java
12/06/00 04:33p <DIR> images
12/07/99 04:25p 7,139 InputDialog.java
12/07/99 04:25p 10,419 LaunchFile.java
12/07/99 04:25p 1,120 Makefile
12/07/99 04:25p 5,151 MessageBoxOnTop.java
12/07/99 04:25p 15,865 NewFileDialog.java
12/07/99 04:25p 10,893 ProgressDialog.java
12/07/99 04:25p 6,824 PropertiesDialog.java
12/07/99 04:25p 41,781 SearchPane.java
12/07/99 04:25p 5,214 SearchPane.resources
12/07/99 04:25p 11,054 SelectCabinetDialog.java
12/07/99 04:25p 10,326 SelectCabinetDialog.resources
12/07/99 04:25p 3,472 SplashScreen.java
12/07/99 04:25p 235,548 SplashScreen.resources
12/07/99 04:25p 1,805 SysImageList.java

```

```

                                test
12/07/99  04:25p                7,510 TreeWatcher.java
                                597,633 bytes
33 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\browser\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:18p                1,610 Entries
12/16/99  06:18p                16 Entries.Log
12/16/99  06:18p                55 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                7 File(s)        2,573 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\browser\images

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:25p                9,484 FILECOPY.AVI
12/07/99  04:25p                1,238 HPLogo.bmp
12/07/99  04:25p                766 LCabClose.ICO
12/07/99  04:25p                766 LCabOpen.ICO
12/07/99  04:25p                766 LFile.ICO
12/07/99  04:25p                766 LFLDClose.ICO
12/07/99  04:25p                766 LFLDOpen.ICO
12/07/99  04:25p                318 SCabClose.ICO
12/07/99  04:25p                318 SCabOpen.ICO
12/07/99  04:25p                20,796 SEARCH.AVI
12/07/99  04:25p                318 SFile.ICO
12/07/99  04:25p                318 SFLDClose.ICO
12/07/99  04:25p                318 SFLDOpen.ICO
                                16 File(s)        36,938 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\browser\images\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:18p                634 Entries
12/16/99  06:18p                62 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                6 File(s)        1,588 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\cl

test

ientapi

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:25p 707 Makefile
12/07/99 04:25p 2,021 NoFileStoreFoundException.java
12/07/99 04:25p 5,825 VFSFileStoreContract.java
12/07/99 04:25p 26,093 VFSFileCabinet.java
12/07/99 04:25p 8,312 VFSFileStoreVocabulary.java
12/07/99 04:25p 10,222 VFSFileStoreFinder.java
12/07/99 04:25p 1,969 VFSFileStoreElement.java
12/07/99 04:25p 4,040 VFSFileStoreDescription.java
12/07/99 04:25p 26,604 VFSFolder.java
12/07/99 04:25p 10,343 VFSResourceVocabulary.java
12/07/99 04:25p 5,781 VFSResourceContract.java
12/07/99 04:25p 28,252 VFSWorkspace.java
15 File(s) 130,169 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\clientapi\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 712 Entries
12/16/99 06:18p 57 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
6 File(s) 1,661 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 48 Entries
12/16/99 06:18p 95 Entries.Log
12/16/99 06:18p 47 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
7 File(s) 1,082 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\intf

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
```



```

12/07/99 04:25p      test
12/07/99 04:25p      556 Makefile
12/07/99 04:25p      5,670 VFSFileIntf.java
12/07/99 04:25p      1,849 VFSFileIntfMessageRegistry.java
12/07/99 04:25p     10,400 VFSFileStub.java
12/07/99 04:25p      8,982 VFSFileStoreStub.java
12/07/99 04:25p      2,223 VFSFileStoreIntfMessageRegistry
.java
12/07/99 04:25p      4,145 VFSFileStoreIntf.java
      10 File(s)      33,825 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\in
tf\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:18p      411 Entries
12/16/99 06:18p      52 Repository
12/16/99 06:18p      46 Root
12/16/99 06:18p      846 Template
      6 File(s)      1,355 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\se
rver

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:25p      9,327 DumpVFSEntries.java
12/15/99 04:00p      542 Makefile
12/07/99 04:25p      5,319 StartFileStore.java
12/07/99 04:25p     10,836 VFSFileStoreService.java
12/07/99 04:25p     25,820 VFSFileStoreImpl.java
12/07/99 04:25p     13,228 VFSFileImpl.java
      9 File(s)      65,072 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\se
rver\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:18p      332 Entries
12/16/99 06:18p      54 Repository
12/16/99 06:18p      46 Root
12/16/99 06:18p      846 Template
      6 File(s)      1,278 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\sh
ell

test

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:25p 967 Makefile
12/07/99 04:25p 3,469 VFScatCommand.java
12/07/99 04:25p 3,830 VFScdCommand.java
12/07/99 04:25p 20,179 VFSCCommand.java
12/07/99 04:25p 10,028 VFScpCommand.java
12/07/99 04:25p 3,111 VFSechoCommand.java
12/07/99 04:25p 7,278 VFSexecCommand.java
12/07/99 04:25p 5,660 VFSexportCommand.java
12/07/99 04:25p 12,226 VFSfindCommand.java
12/07/99 04:25p 8,167 VFSimportCommand.java
12/07/99 04:25p 10,912 VFSlsCommand.java
12/07/99 04:25p 5,754 VFSmkdirCommand.java
12/07/99 04:25p 6,212 VFSmkfileCommand.java
12/07/99 04:25p 14,992 VFSmvCommand.java
12/07/99 04:25p 2,901 VFSpwdCommand.java
12/07/99 04:25p 4,576 VFSresetCommand.java
12/07/99 04:25p 10,724 VFSrmCommand.java
12/07/99 04:25p 6,855 VFSrmdirCommand.java
12/07/99 04:25p 5,909 VFSsetCommand.java
12/07/99 04:25p 5,986 VFSshareCommand.java
12/07/99 04:25p 1,776 VFSShellException.java
12/07/99 04:25p 15,496 VFSShell.java
12/07/99 04:25p 7,593 VFSshowCommand.java
12/07/99 04:25p 1,775 VFSStackException.java
12/07/99 04:25p 3,280 VFSStack.java
28 File(s) 179,656 bytes
Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\shell\CVS

```

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 1,379 Entries
12/16/99 06:18p 53 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
6 File(s) 2,324 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\util

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS

```

```

                                test
12/07/99  04:25p                401 Makefile
12/07/99  04:25p                3,469 VFSStrings.java
                                3,870 bytes
5 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\contrib\vfs\src\vfs\util\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:18p                                100 Entries
12/16/99  06:18p                                52 Repository
12/16/99  06:18p                                46 Root
12/16/99  06:18p                                846 Template
6 File(s)                                1,044 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:17p                                431 Entries
12/16/99  06:22p                                110 Entries.Log
12/16/99  06:17p                                27 Repository
12/16/99  06:17p                                46 Root
12/16/99  06:17p                                846 Template
7 File(s)                                1,460 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\extern

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/06/00  04:33p                <DIR>                CVS
12/06/00  04:33p                <DIR>                ldap
12/06/00  04:33p                <DIR>                openxml
12/06/00  04:33p                <DIR>                oracle-lib
6 File(s)                                0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\extern\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:18p                                3 Entries
12/16/99  06:18p                                51 Entries.Log
12/16/99  06:18p                                34 Repository
12/16/99  06:18p                                46 Root
12/16/99  06:18p                                846 Template
7 File(s)                                980 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\extern\ldap

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/08/99 01:52p 160,475 ldapjdk.jar
4 File(s) 160,475 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\extern\ldap\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 51 Entries
12/16/99 06:18p 39 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
6 File(s) 982 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\extern\openxml

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/08/99 01:49p 357,071 openxml.jar
4 File(s) 357,071 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\extern\openxml\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 51 Entries
12/16/99 06:18p 42 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
6 File(s) 985 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\extern\oracle-lib

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/08/99 01:57p 800,174 classes111.zip
12/06/00 04:33p <DIR> CVS
4 File(s) 800,174 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\extern\oracle-lib\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 54 Entries
```

		test	
12/16/99	06:18p		45 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
		6 File(s)	991 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	echo
12/06/00	04:33p	<DIR>	ESChat
12/07/99	04:25p		290 Makefile
12/06/00	04:33p	<DIR>	ManagedEcho
12/06/00	04:33p	<DIR>	ManagedPrintServer
12/06/00	04:33p	<DIR>	PrintServer
		9 File(s)	290 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		48 Entries
12/16/99	06:19p		100 Entries.Log
12/16/99	06:18p		35 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
		7 File(s)	1,075 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\echo

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	config
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		4,998 echo.vjp
12/07/99	04:25p		4,127 README.txt
12/06/00	04:33p	<DIR>	src
		7 File(s)	9,125 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\config

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		25 col.MYCO
12/07/99	04:25p		26 co2.MYCO
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	multicore

```

                                test
12/06/00  04:33p                <DIR>                singlecore
                                7 File(s)              51 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\config\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:19p                                93 Entries
12/16/99  06:19p                                39 Entries.Log
12/16/99  06:19p                                47 Repository
12/16/99  06:19p                                46 Root
12/16/99  06:19p                                846 Template
                                7 File(s)              1,071 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\config\multicore

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/07/99  04:25p                                64 client.prop
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:25p                                744 EchoClient.ini
12/07/99  04:25p                                730 EchoServer.ini
12/07/99  04:25p                                42 server.prop
                                7 File(s)              1,580 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\config\multicore\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:19p                                201 Entries
12/16/99  06:19p                                57 Repository
12/16/99  06:19p                                46 Root
12/16/99  06:19p                                846 Template
                                6 File(s)              1,150 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\config\singlecore

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/07/99  04:25p                                60 client.prop
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:25p                                199 EchoClient.ini
12/07/99  04:25p                                328 EchoServer.ini
12/07/99  04:25p                                60 server.prop

```

test
7 File(s) 647 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\config\singlecore\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 201 Entries
12/16/99 06:19p 58 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
6 File(s) 1,151 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 95 Entries
12/16/99 06:19p 29 Entries.Log
12/16/99 06:19p 40 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
7 File(s) 1,056 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\src

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:25p 5,216 EchoServer.java
12/07/99 04:25p 6,205 EchoClient.java
12/07/99 04:25p 2,397 EchoServiceIntf.esidl
12/07/99 04:25p 3,838 EchoServiceStub.java
12/07/99 04:25p 1,388 EchoServiceIntfMessageRegistry.
java
12/07/99 04:25p 2,397 EchoServiceIntf.java
12/07/99 04:25p 2,103 EchoServiceImpl.java
12/07/99 04:25p 22,016 echoUML.doc
12/07/99 04:25p 661 Makefile
12 File(s) 46,221 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\echo\src\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 501 Entries
12/16/99 06:19p 44 Repository
12/16/99 06:19p 46 Root
```

```

                                test
12/16/99  06:19p                846 Template
                                1,437 bytes
        6 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      config
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p                3,550 README.txt
12/06/00  04:33p      <DIR>      src
        6 File(s)                3,550 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\config

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      multicore
12/06/00  04:33p      <DIR>      singlecore
        5 File(s)                0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\config\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                3 Entries
12/16/99  06:18p                39 Entries.Log
12/16/99  06:18p                49 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
        7 File(s)                983 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\config\multicore

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p                58 client.pr
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p                757 ESChatServer.ini
12/07/99  04:25p                164 ESChatClient.ini
12/07/99  04:25p                570 ESChatAddCore.ini
12/07/99  04:25p                58 server.pr
        8 File(s)                1,607 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\config\

test

multicore\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 255 Entries
12/16/99 06:18p 59 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
6 File(s) 1,206 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\config\singlecore

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:25p 57 client.pr
12/06/00 04:33p <DIR> CVS
12/07/99 04:25p 740 ESChatServer.ini
12/07/99 04:25p 204 ESChatClient.ini
12/07/99 04:25p 57 server.pr
7 File(s) 1,058 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\config\singlecore\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 201 Entries
12/16/99 06:18p 60 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
6 File(s) 1,153 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:18p 50 Entries
12/16/99 06:18p 29 Entries.Log
12/16/99 06:18p 42 Repository
12/16/99 06:18p 46 Root
12/16/99 06:18p 846 Template
7 File(s) 1,013 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\src

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
```

```

12/07/99  04:25p          test
12/06/00  04:33p          3,466 ChatEventDist.java
12/07/99  04:25p      <DIR>      CVS
12/07/99  04:25p          23,040 ESChatUML.doc
12/07/99  04:25p          9,527 ESChat.java
12/07/99  04:25p          404 Makefile
12/07/99  04:25p          3,219 MessageDialog.java
12/07/99  04:25p          3,185 SubscriberImpl.java
          9 File(s)          42,841 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ESChat\src\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p          312 Entries
12/16/99  06:18p          46 Repository
12/16/99  06:18p          46 Root
12/16/99  06:18p          846 Template
          6 File(s)          1,250 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedEcho

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      config
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p          2,236 README.txt
12/06/00  04:33p      <DIR>      src
          6 File(s)          2,236 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedEcho\config

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p          25 col.MYCO
12/07/99  04:25p          26 co2.MYCO
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p          443 ManagedEcho.ini
12/07/99  04:25p          108 testMgr.ini
          7 File(s)          602 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedEcho\config\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p          193 Entries
12/16/99  06:18p          54 Repository

```

```

                                test
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                1,139 bytes
        6 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedEcho\CV
S

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/16/99  06:18p                50 Entries
12/16/99  06:18p                29 Entries.Log
12/16/99  06:18p                47 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
        7 File(s)                1,018 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedEcho\sr
c

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/06/00  04:33p                <DIR>      CVS
12/07/99  04:25p                2,212 EchoServiceIntf.esidl
12/07/99  04:25p                2,343 EchoServiceStub.java
12/07/99  04:25p                1,408 EchoServiceIntfMessageRegistry.
java
12/07/99  04:25p                2,216 EchoServiceIntf.java
12/07/99  04:25p                2,154 EchoServiceImpl.java
12/07/99  04:25p                688 Makefile
12/07/99  04:25p                6,077 ManagedEcho.java
12/07/99  04:25p                2,632 startManagedEcho.java
12/07/99  04:25p                4,196 testMgr.java
        12 File(s)                23,926 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedEcho\sr
c\CVS

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/16/99  06:18p                509 Entries
12/16/99  06:18p                51 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
        6 File(s)                1,452 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      config
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      doc
12/09/99  06:44a      5,800 README.txt
12/06/00  04:33p      <DIR>      SampleFiles
12/06/00  04:33p      <DIR>      src
                                8 File(s)
                                5,800 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\config

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      multicore
12/07/99  04:25p      380 Print.ini
12/06/00  04:33p      <DIR>      singlecore
                                6 File(s)
                                380 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\config\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p      49 Entries
12/16/99  06:18p      39 Entries.Log
12/16/99  06:18p      61 Repository
12/16/99  06:18p      46 Root
12/16/99  06:18p      846 Template
                                7 File(s)
                                1,041 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\config\multicore

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p      42 Client.prop
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p      341 PC.ini
12/07/99  04:25p      538 PC-Core.ini
12/09/99  07:02a      1,696 PS.ini
                                7 File(s)
                                2,617 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\config\multicore\CVS

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                      185 Entries
12/16/99  06:18p                      71 Repository
12/16/99  06:18p                      46 Root
12/16/99  06:18p                      846 Template
                                6 File(s)      1,148 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\config\singlecore

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p                      42 Client.prop
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p                      220 PC.ini
12/07/99  04:25p                      376 print.ini
12/09/99  07:03a                      1,278 PS.ini
12/07/99  04:25p                      42 Server.prop
                                8 File(s)      1,958 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\config\singlecore\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                      231 Entries
12/16/99  06:18p                      72 Repository
12/16/99  06:18p                      46 Root
12/16/99  06:18p                      846 Template
                                6 File(s)      1,195 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                      50 Entries
12/16/99  06:18p                      63 Entries.Log
12/16/99  06:18p                      54 Repository
12/16/99  06:18p                      46 Root
12/16/99  06:18p                      846 Template
                                7 File(s)      1,059 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\doc

```

12/06/00  04:33p      <DIR>      .

```

```

                                test
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/09/99  07:01a      96,647 Userguide.doc
                        4 File(s)    96,647 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\doc\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p      53 Entries
12/16/99  06:18p      58 Repository
12/16/99  06:18p      46 Root
12/16/99  06:18p      846 Template
                        6 File(s)    1,003 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\SampleFiles

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p      181,760 test.doc
12/07/99  04:25p      351 test.ini
12/07/99  04:25p      6 test.ppt
12/07/99  04:25p      49 test.txt
12/07/99  04:25p      6 test.xls
                        8 File(s)    182,172 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\SampleFiles\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p      228 Entries
12/16/99  06:18p      66 Repository
12/16/99  06:18p      46 Root
12/16/99  06:18p      846 Template
                        6 File(s)    1,186 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\src

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/15/99  04:00p      799 Makefile
12/07/99  04:25p      10,298 ManagedPrintService.java

```

```

                                test
12/07/99  04:25p                6,133 PolicyWriter.java
12/07/99  04:25p               18,611 PrintClient.java
12/07/99  04:25p                3,805 PrinterStub.java
12/07/99  04:25p                1,884 PrinterIntf.java
12/07/99  04:25p               14,133 PrinterImpl.java
12/07/99  04:25p                5,539 PrintJobInfo.java
12/07/99  04:25p               18,441 PrintServer.java
                                79,643 bytes
                                12 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\samples\ManagedPrintSe
rver\src\CVS

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/16/99  06:18p                482 Entries
12/16/99  06:18p                58 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p               846 Template
                                1,432 bytes
                                6 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/06/00  04:33p                <DIR>      config
12/06/00  04:33p                <DIR>      CVS
12/07/99  04:25p               4,499 README.txt
12/06/00  04:33p                <DIR>      SampleFiles
12/06/00  04:33p                <DIR>      src
                                4,499 bytes
                                7 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\co
nfig

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/06/00  04:33p                <DIR>      col
12/06/00  04:33p                <DIR>      co2
12/06/00  04:33p                <DIR>      CVS
12/06/00  04:33p                <DIR>      multicore
12/07/99  04:25p                297 Print.ini
12/07/99  04:25p               1,973 printer.xml
12/07/99  04:25p               1,688 printer1.xml
12/07/99  04:25p               1,760 printit.pl
12/07/99  04:25p               1,571 printVocab.xml
12/06/00  04:33p                <DIR>      singlecore
                                7,289 bytes
                                12 File(s)

```

test

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\col

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		25 col.MYCO
12/06/00	04:33p	<DIR>	CVS
		4 File(s)	25 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\col\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		48 Entries
12/16/99	06:18p		58 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
		6 File(s)	998 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\co2

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:25p		25 co2.MYCO
12/06/00	04:33p	<DIR>	CVS
		4 File(s)	25 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\co2\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		48 Entries
12/16/99	06:18p		58 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
		6 File(s)	998 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		244 Entries
12/16/99	06:18p		65 Entries.Log
12/16/99	06:18p		54 Repository


```

                                test
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                1,255 bytes
7 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\multicore

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p                42 Client.prop
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p                202 PC.ini
12/07/99  04:25p                557 PC-Core.ini
12/07/99  04:25p                817 PS.ini
7 File(s)                1,618 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\multicore\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                185 Entries
12/16/99  06:18p                64 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
6 File(s)                1,141 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\singlecore

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:25p                42 Client.prop
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:25p                184 PC.ini
12/07/99  04:25p                426 PS.ini
6 File(s)                652 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\config\singlecore\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:18p                137 Entries
12/16/99  06:18p                65 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
6 File(s)                1,094 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\CV
S

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		50 Entries
12/16/99	06:18p		50 Entries.Log
12/16/99	06:18p		47 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
7 File(s)			1,039 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\Sa
mpleFiles

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:25p		181,760 test.doc
12/07/99	04:25p		351 test.ini
12/07/99	04:25p		6 test.ppt
12/07/99	04:25p		49 test.txt
12/07/99	04:25p		6 test.xls
8 File(s)			182,172 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\Sa
mpleFiles\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:18p		228 Entries
12/16/99	06:18p		59 Repository
12/16/99	06:18p		46 Root
12/16/99	06:18p		846 Template
6 File(s)			1,179 bytes

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\sr
c

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/15/99	04:00p		995 Makefile
12/07/99	04:25p		6,152 PrintServer.java
12/07/99	04:25p		16,750 PrintClient.java
12/07/99	04:25p		2,375 PrintJobInfo.esidl
12/07/99	04:25p		2,524 PrintJobInfo.java

```

                                test
12/07/99  04:25p                60,928 PrintServerUML.doc
12/07/99  04:25p                2,343 PrintServiceIntf.esidl
12/07/99  04:25p                2,958 PrintServiceStub.java
12/07/99  04:25p                1,574 PrintServiceIntfMessageRegistry
.java
12/07/99  04:25p                2,347 PrintServiceIntf.java
12/07/99  04:25p                6,187 PrintServiceImpl.java
                                14 File(s)          105,133 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\samples\PrintServer\src\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:19p                624 Entries
12/16/99  06:18p                51 Repository
12/16/99  06:18p                46 Root
12/16/99  06:18p                846 Template
                                6 File(s)          1,567 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ...
12/06/00  04:33p                <DIR>                c
12/06/00  04:33p                <DIR>                CVS
12/06/00  04:33p                <DIR>                java
12/06/00  04:33p                <DIR>                perl
12/06/00  04:33p                <DIR>                python
                                7 File(s)          0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\c

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/06/00  04:33p                <DIR>                cesi
12/06/00  04:33p                <DIR>                CVS
                                4 File(s)          0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\c\cesi

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/07/99  04:25p                49,436 abi.c
12/07/99  04:25p                16,525 abi.h
12/07/99  04:25p                15,614 abi_base.c
12/07/99  04:25p                4,223 abi_base.h
12/06/00  04:33p                <DIR>                CVS

```

```

                                test
12/07/99  04:25p                19,827 eslib.c
12/07/99  04:25p                3,512 eslib.h
12/07/99  04:25p                 620 makefile
12/07/99  04:25p                 631 makefile_w32
12/07/99  04:25p                 408 README.txt
12/07/99  04:25p                3,716 rtc.c
12/07/99  04:25p                8,036 rts.c
12/07/99  04:25p                3,745 tc.c
12/07/99  04:25p                5,654 ts.c
12/07/99  04:25p               13,071 wservpipe.c
                                17 File(s)
                                145,018 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\c\cesi\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                624 Entries
12/16/99  06:19p                38 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p               846 Template
                                6 File(s)
                                1,554 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\c\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                3 Entries
12/16/99  06:19p               14 Entries.Log
12/16/99  06:19p               33 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p               846 Template
                                7 File(s)
                                942 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                3 Entries
12/16/99  06:22p               55 Entries.Log
12/16/99  06:19p               31 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p               846 Template
                                7 File(s)
                                981 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..

```

```

                                test
12/06/00  04:33p          <DIR>          CVS
12/07/99  04:25p                                187 Makefile
12/06/00  04:33p          <DIR>          net
                                5 File(s)          187 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\CVS

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/16/99  06:19p                                48 Entries
12/16/99  06:19p                                13 Entries.Log
12/16/99  06:19p                                36 Repository
12/16/99  06:19p                                46 Root
12/16/99  06:19p                                846 Template
                                7 File(s)          989 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/06/00  04:33p          <DIR>          CVS
12/06/00  04:33p          <DIR>          espeak
                                4 File(s)          0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\CVS

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/16/99  06:19p                                3 Entries
12/16/99  06:19p                                16 Entries.Log
12/16/99  06:19p                                40 Repository
12/16/99  06:19p                                46 Root
12/16/99  06:19p                                846 Template
                                7 File(s)          951 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/06/00  04:33p          <DIR>          CVS
12/06/00  04:33p          <DIR>          infra
12/06/00  04:33p          <DIR>          jesi
12/07/99  04:25p                                477 Makefile
12/06/00  04:33p          <DIR>          services
12/06/00  04:33p          <DIR>          util
12/07/99  04:25p                                2,099 Version.java
12/06/00  04:33p          <DIR>          webaccess
                                10 File(s)          2,576 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\CV
S

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:19p		97 Entries
12/16/99	06:21p		80 Entries.Log
12/16/99	06:19p		47 Repository
12/16/99	06:19p		46 Root
12/16/99	06:19p		846 Template
7 File(s)			1,116 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\in
fra

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	cci
12/06/00	04:33p	<DIR>	client
12/06/00	04:33p	<DIR>	core
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	intercorecom
2/07/99	04:25p		269 Makefile
2/06/00	04:33p	<DIR>	xml
9 File(s)			269 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\in
fra\cci

2/06/00	04:33p	<DIR>	.
2/06/00	04:33p	<DIR>	..
2/06/00	04:33p	<DIR>	coreapi
12/06/00	04:33p	<DIR>	coreproxy
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	events
12/06/00	04:33p	<DIR>	exception
12/06/00	04:33p	<DIR>	export
12/07/99	04:25p		357 Makefile
12/06/00	04:33p	<DIR>	management
12/06/00	04:33p	<DIR>	messaging
12/06/00	04:33p	<DIR>	metadata
12/06/00	04:33p	<DIR>	naming
12/06/00	04:33p	<DIR>	security
14 File(s)			357 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\in
fra\cci\coreapi

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:25p 6,485 CoreManagementInterface.java
12/07/99 04:25p 8,572 CoreNames.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:25p 8,626 ImporterExporterInterface.java
12/07/99 04:25p 2,698 KeyInterface.java
12/07/99 04:25p 4,645 KeyRingInterface.java
12/07/99 04:25p 5,388 MailboxInterface.java
12/07/99 04:25p 805 Makefile
12/07/99 04:25p 9,933 NameFrameInterface.java
12/07/99 04:25p 5,301 ProtectionDomainInterface.java
12/07/99 04:25p 6,118 RepositoryViewInterface.java
12/07/99 04:25p 24,869 ResourceManipulationInterface.j
ava
12/07/99 04:25p 3,754 ResourceFactoryInterface.java
12/07/99 04:25p 3,284 ResourceContractInterface.java
12/07/99 04:25p 4,752 SystemMonitorInterface.java
12/07/99 04:25p 5,108 VocabularyToolboxInterface.java
12/07/99 04:25p 4,922 VocabularyInterface.java
19 File(s) 105,260 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\coreapi\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 990 Entries
12/16/99 06:19p 65 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
6 File(s) 1,947 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\coreproxy

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 6,240 ConnectionFactoryInterface.java
12/07/99 04:26p 4,241 CoreProxyInterface.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 400 Makefile
6 File(s) 10,881 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\coreproxy\CVS

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                      176 Entries
12/16/99  06:19p                      67 Repository
12/16/99  06:19p                      46 Root
12/16/99  06:19p                      846 Template
                                6 File(s)          1,135 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                      48 Entries
12/16/99  06:19p                      178 Entries.Log
12/16/99  06:19p                      57 Repository
12/16/99  06:19p                      46 Root
12/16/99  06:19p                      846 Template
                                7 File(s)          1,175 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\events

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p                      6,417 CoreEvent.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                      5,705 EventAttributeSet.java
12/07/99  04:26p                      7,891 EventList.java
12/07/99  04:26p                      14,456 Event.java
12/07/99  04:26p                      413 Makefile
                                8 File(s)          34,882 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\events\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                      256 Entries
12/16/99  06:19p                      64 Repository
12/16/99  06:19p                      46 Root
12/16/99  06:19p                      846 Template
                                6 File(s)          1,212 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\exception

```

12/06/00  04:33p      <DIR>      .

```


12/06/00	04:33p		test
12/07/99	04:26p	<DIR>	..
12/06/00	04:33p		2,419 CorePanicException.java
12/07/99	04:26p	<DIR>	CVS
12/07/99	04:26p		2,067 EmptyMappingException.java
12/07/99	04:26p		8,082 ESEExceptionInfo.java
12/07/99	04:26p		8,668 ESEException.java
12/07/99	04:26p		2,404 ESInvocationException.java
12/07/99	04:26p		1,839 ESLibException.java
12/07/99	04:26p		2,356 ESNameFrameException.java
12/07/99	04:26p		2,375 ESRemoteException.java
12/07/99	04:26p		7,431 ESRuntimeException.java
12/07/99	04:26p		2,555 ESServiceException.java
12/07/99	04:26p		2,458 ExportFailedException.java
12/07/99	04:26p		2,458 ImportFailedException.java
12/07/99	04:26p		2,123 InvalidValueException.java
12/07/99	04:26p		2,092 InvalidTypeException.java
12/07/99	04:26p		4,432 InvalidParameterException.java
12/07/99	04:26p		2,204 InvalidNameException.java
12/07/99	04:26p		2,158 LookupFailedException.java
12/07/99	04:26p		1,585 Makefile
12/07/99	04:26p		2,454 MethodNotImplementedException.j
12/07/99	04:26p		2,204 MultipleResolvedBindingExceptio
12/07/99	04:26p		2,094 NameCollisionException.java
12/07/99	04:26p		2,064 NameNotFoundException.java
12/07/99	04:26p		2,143 NamingException.java
12/07/99	04:26p		2,016 NotExportableException.java
12/07/99	04:26p		2,022 NotImportableException.java
12/07/99	04:26p		2,152 NullParameterException.java
12/07/99	04:26p		2,642 OutofOrderRequestException.java
12/07/99	04:26p		2,117 PartialStateUpdateException.jav
12/07/99	04:26p		2,119 PermissionDeniedException.java
12/07/99	04:26p		2,066 QuotaExhaustedException.java
12/07/99	04:26p		2,035 RecoverableDeliveryException.ja
12/07/99	04:26p		2,274 RecoverableCoreException.java
12/07/99	04:26p		2,088 RepositoryFullException.java
12/07/99	04:26p		2,118 RequestNotDeliveredException.ja
12/07/99	04:26p		2,453 ServicePanicException.java
12/07/99	04:26p		2,164 StaleEntryAccessException.java
12/07/99	04:26p		2,112 TimedOutException.java
12/07/99	04:26p		2,332 UndeliverableRequestException.j
12/07/99	04:26p		2,216 UnrecoverableDeliveryException.

test
12/07/99 04:26p 2,009 UnresolvedBindingException.java
43 File(s) 107,600 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\exception\CVS

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 2,545 Entries
12/16/99 06:19p 67 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
6 File(s) 3,504 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\export

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 15,693 ExportFE.java
12/07/99 04:26p 27,068 ExportContext.java
12/07/99 04:26p 17,102 ImportContext.java
12/07/99 04:26p 397 Makefile
7 File(s) 60,260 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\export\CVS

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 208 Entries
12/16/99 06:19p 64 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
6 File(s) 1,164 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\management

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 368 Makefile
12/07/99 04:26p 3,081 ManagedServiceOpcodes.java
5 File(s) 3,449 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\management\CVS

test

fra\cci\management\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p      111 Entries
12/16/99 06:19p      68 Repository
12/16/99 06:19p      46 Root
12/16/99 06:19p     846 Template
                6 File(s)      1,071 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\in
fra\cci\messaging

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p      4,702 BootstrapReply.java
12/07/99 04:26p     17,492 ChannelWriter.java
12/07/99 04:26p     23,263 ChannelReader.java
12/07/99 04:26p     20,951 Channel.java
12/07/99 04:26p     12,280 ClientChannel.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p      3,284 ESSerializable.java
12/07/99 04:26p      6,155 FIFOQueue.java
12/07/99 04:26p     13,488 InboxMessageAtom.java
12/07/99 04:26p      9,259 IVMChannel.java
12/07/99 04:26p      7,410 IVMQueue.java
12/07/99 04:26p      2,697 IVMRendezvous.java
12/07/99 04:26p      962 Makefile
12/07/99 04:26p      7,804 Message.java
12/07/99 04:26p     14,287 MessageAtom.java
12/07/99 04:26p     34,717 MessageRegistry.java
12/07/99 04:26p     22,555 MessageOutputStream.java
12/07/99 04:26p     22,914 MessageInputStream.java
12/07/99 04:26p      6,647 MessageBufferedStream.java
12/07/99 04:26p      4,358 MsgFilter.java
12/07/99 04:26p     12,195 OutboxMessageAtom.java
12/07/99 04:26p      4,369 PayloadReference.java
12/07/99 04:26p      4,380 PayloadFromCore.java
12/07/99 04:26p      3,467 PayloadForCore.java
12/07/99 04:26p      3,869 Rendezvous.java
12/07/99 04:26p      6,153 ResourceInfo.java
12/07/99 04:26p      3,800 SystemMonitorHelper.java
12/07/99 04:26p     12,380 TCPChannel.java
12/07/99 04:26p      3,229 TCPRendezvous.java
                31 File(s)    289,067 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\in
fra\cci\messaging\CVS

test

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 1,538 Entries
12/16/99 06:19p 67 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
6 File(s) 2,497 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\metadata

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 12,656 Attribute.java
12/07/99 04:26p 15,404 AttributeSet.java
12/07/99 04:26p 9,786 AttributePropertySet.java
12/07/99 04:26p 21,375 AttributeProperty.java
12/07/99 04:26p 10,960 AttributePredicate.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 29,182 ESUID.java
12/07/99 04:26p 704 Makefile
12/07/99 04:26p 1,756 MatcherInterface.java
12/07/99 04:26p 3,760 NamedObject.java
12/07/99 04:26p 1,759 PromoterInterface.java
12/07/99 04:26p 31,730 ResourceSpecification.java
12/07/99 04:26p 6,603 ResourceDescription.java
12/07/99 04:26p 8,575 RSD.java
12/07/99 04:26p 15,618 SearchRecipe.java
12/07/99 04:26p 6,406 SearchPredicate.java
12/07/99 04:26p 20,342 ValueType.java
12/07/99 04:26p 59,374 Value.java
20 File(s) 255,990 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\metadata\CVS

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 929 Entries
12/16/99 06:19p 66 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
6 File(s) 1,887 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\naming

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p                                3,069 Accessor.java
12/06/00  04:33p      <DIR>                                CVS
12/07/99  04:26p                                27,188 ESName.java
12/07/99  04:26p                                7,868 LiteralName.java
12/07/99  04:26p                                441 Makefile
12/07/99  04:26p                                7,453 NameSearchPolicy.java
12/07/99  04:26p                                4,852 ResourceReference.java
                                9 File(s)                50,871 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\naming\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                                316 Entries
12/16/99  06:19p                                64 Repository
12/16/99  06:19p                                46 Root
12/16/99  06:19p                                846 Template
                                6 File(s)                1,272 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\security

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                                2,228 Lockable.java
12/07/99  04:26p                                6,335 LockedPermissions.java
12/07/99  04:26p                                4,131 Lock.java
12/07/99  04:26p                                394 Makefile
                                7 File(s)                13,088 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\cci\security\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                                203 Entries
12/16/99  06:19p                                66 Repository
12/16/99  06:19p                                46 Root
12/16/99  06:19p                                846 Template
                                6 File(s)                1,161 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      clientapi
12/06/00  04:33p      <DIR>      coreproxy
12/06/00  04:33p      <DIR>      corestubs
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      exception
12/06/00  04:33p      <DIR>      impl
12/07/99  04:26p      301 Makefile
12/06/00  04:33p      <DIR>      util
                                10 File(s)
                                301 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\clientapi

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p      2,613 ESContractIntf.java
12/07/99  04:26p      3,793 ESInboxIntf.java
12/07/99  04:26p      2,216 ESIntf.java
12/07/99  04:26p      3,218 ESKeyIntf.java
12/07/99  04:26p      3,368 ESKeyRingIntf.java
12/07/99  04:26p      10,319 ESNameFrameIntf.java
12/07/99  04:26p      3,936 ESProtectionDomainIntf.java
12/07/99  04:26p      3,788 ESRemoteObjIntf.java
12/07/99  04:26p      13,674 ESRLIntf.java
12/07/99  04:26p      3,361 ESServiceIntf.java
12/07/99  04:26p      40,793 ESShellIntf.java
12/07/99  04:26p      3,018 ESSystemMonitorIntf.java
12/07/99  04:26p      3,339 ESViewIntf.java
12/07/99  04:26p      3,267 ESVocabularyIntf.java
12/07/99  04:26p      674 Makefile
                                18 File(s)
                                101,377 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\clientapi\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p      818 Entries
12/16/99  06:19p      70 Repository
12/16/99  06:19p      46 Root
12/16/99  06:19p      846 Template
                                6 File(s)
                                1,780 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\coreproxy

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 33,299 ESConnectionFactory.java
12/07/99 04:26p 7,074 ESCoreProxy.java
12/07/99 04:26p 390 Makefile
        6 File(s) 40,763 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\coreproxy\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 162 Entries
12/16/99 06:19p 70 Repository
12/16/99 06:19p 46 Root
12/16/99 06:19p 846 Template
        6 File(s) 1,124 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\corestubs

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 12,959 ImporterExporter.java
12/07/99 04:26p 8,599 InterfaceToCore.java
12/07/99 04:26p 5,527 KeyRing.java
12/07/99 04:26p 3,281 Key.java
12/07/99 04:26p 7,386 Mailbox.java
12/07/99 04:26p 642 Makefile
12/07/99 04:26p 17,945 NameFrame.java
12/07/99 04:26p 6,081 ProtectionDomain.java
12/07/99 04:26p 9,201 RepositoryView.java
12/07/99 04:26p 42,410 ResourceManipulation.java
12/07/99 04:26p 4,920 ResourceFactory.java
12/07/99 04:26p 3,469 ResourceContract.java
12/07/99 04:26p 5,250 SystemMonitor.java
12/07/99 04:26p 5,396 Vocabulary.java
        17 File(s) 133,066 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\corestubs\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:19p 755 Entries
```

```

                                test
12/16/99  06:19p                70 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                6 File(s)          1,717 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:19p                48 Entries
12/16/99  06:19p                104 Entries.Log
12/16/99  06:19p                60 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                7 File(s)          1,104 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\exception

```

02/06/00  04:33p                <DIR>          .
02/06/00  04:33p                <DIR>          ..
02/07/99  04:26p                2,505 CoreNotFoundException.java
02/06/00  04:33p                <DIR>          CVS
02/07/99  04:26p                409 Makefile
02/07/99  04:26p                2,937 UnexpectedExceptionException.java
                                6 File(s)          5,851 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\exception\CVS

```

02/06/00  04:33p                <DIR>          .
02/06/00  04:33p                <DIR>          ..
12/16/99  06:19p                181 Entries
12/16/99  06:19p                70 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                6 File(s)          1,143 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\impl

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:26p                5,506 ESContract.java
12/07/99  04:26p                9,352 ESCoreManagementService.java

```


		test	
12/07/99	04:26p	17,920	ESImporterExporter.java
12/07/99	04:26p	8,421	ESInbox.java
12/07/99	04:26p	7,753	ESKeyRing.java
12/07/99	04:26p	4,747	ESKey.java
12/07/99	04:26p	17,440	ESManagedCoreManagedResource.java
12/07/99	04:26p	23,204	ESNameFrame.java
12/07/99	04:26p	8,391	ESProtectionDomain.java
12/07/99	04:26p	10,068	ESRemoteObj.java
12/07/99	04:26p	36,740	ESRL.java
12/07/99	04:26p	3,837	ESService.java
12/07/99	04:26p	112,191	ESShell.java
12/07/99	04:26p	6,037	ESSystemMonitor.java
12/07/99	04:26p	13,322	ESView.java
12/07/99	04:26p	9,792	ESVocabulary.java
12/07/99	04:26p	8,090	ES.java
12/07/99	04:26p	738	Makefile
12/07/99	04:26p	16,647	StartService.java
	22 File(s)	320,196 bytes	

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\impl\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:19p	1,011	Entries
12/16/99	06:19p	65	Repository
12/16/99	06:19p	46	Root
12/16/99	06:19p	846	Template
	6 File(s)	1,968 bytes	

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\util

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p	10,093	CoreNameList.java
12/06/00	04:33p		CVS
12/07/99	04:26p	4,571	ESPOPInfo.java
12/07/99	04:26p	2,699	ExtractException.java
12/07/99	04:26p	4,557	HelperMethods.java
12/07/99	04:26p	635	Makefile
12/07/99	04:26p	12,720	MessageState.java
12/07/99	04:26p	16,524	MessageHandler.java
12/07/99	04:26p	11,781	MethodPermissionMap.java
12/07/99	04:26p	5,547	MsgIdentityFilter.java
12/07/99	04:26p	5,919	MsgRSDFilter.java
12/07/99	04:26p	4,811	ParameterList.java

12/07/99	04:26p	test
12/07/99	04:26p	5,598 ParamUnit.java
12/07/99	04:26p	2,340 Trace.java
		5,560 UserInfo.java
	17 File(s)	93,355 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\client\util\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:19p		753 Entries
12/16/99	06:19p		65 Repository
12/16/99	06:19p		46 Root
12/16/99	06:19p		846 Template
	6 File(s)		1,710 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	contract
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	exception
12/06/00	04:33p	<DIR>	export
12/06/00	04:33p	<DIR>	mailbox
12/07/99	04:40p		443 Makefile
12/06/00	04:33p	<DIR>	management
12/06/00	04:33p	<DIR>	naming
12/06/00	04:33p	<DIR>	pd
12/06/00	04:33p	<DIR>	plugins
12/06/00	04:33p	<DIR>	quota
12/06/00	04:33p	<DIR>	repository
12/06/00	04:33p	<DIR>	resource
12/06/00	04:33p	<DIR>	router
12/06/00	04:33p	<DIR>	security
12/06/00	04:33p	<DIR>	startup
12/06/00	04:33p	<DIR>	sysmon
12/06/00	04:33p	<DIR>	util
12/06/00	04:33p	<DIR>	vocabulary
	21 File(s)		443 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\contract

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:40p		12,648 Contract.java

```

                                test
12/06/00  04:33p          <DIR>          CVS
12/07/99  04:40p          375 Makefile
12/07/99  04:40p      2,666 RegistryIF.java
                        6 File(s)      15,689 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\contract\CVS

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/16/99  06:19p          150 Entries
12/16/99  06:19p          67 Repository
12/16/99  06:19p          46 Root
12/16/99  06:19p      846 Template
                        6 File(s)      1,109 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\CVS

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/16/99  06:19p          48 Entries
12/16/99  06:20p      289 Entries.Log
12/16/99  06:19p          58 Repository
12/16/99  06:19p          46 Root
12/16/99  06:19p      846 Template
                        7 File(s)      1,287 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\exception

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/07/99  04:40p      2,531 AssertionError.java
12/06/00  04:33p          <DIR>          CVS
12/07/99  04:40p          372 Makefile
                        5 File(s)      2,903 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\exception\CVS

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/16/99  06:19p          114 Entries
12/16/99  06:19p          68 Repository
12/16/99  06:19p          46 Root
12/16/99  06:19p      846 Template
                        6 File(s)      1,074 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\export

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:40p                5,999 ExportMessageInputStream.java
12/07/99  04:40p                42,803 Exporter.java
12/07/99  04:40p                8,112 FrameMapEntry.java
12/07/99  04:40p               19,363 ImportExportNames.java
12/07/99  04:40p               26,949 ImporterExporter.java
12/07/99  04:40p               55,249 Importer.java
12/07/99  04:40p                485 Makefile
          10 File(s)          158,960 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\export\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                386 Entries
12/16/99  06:19p                65 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p               846 Template
          6 File(s)          1,343 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\mailbox

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:40p                6,974 ChannelWriterThread.java
12/07/99  04:40p                2,655 ChannelWriterDaemon.java
12/07/99  04:40p                6,985 ChannelThreadPool.java
12/07/99  04:40p                4,675 CoreChannel.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:40p               10,995 Inbox.java
12/07/99  04:40p                527 Makefile
12/07/99  04:40p                2,344 MessageQueueItem.java
12/07/99  04:40p                5,321 MessageQueue.java
12/07/99  04:40p               10,818 Outbox.java
          12 File(s)          51,294 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\mailbox\CVS

```
12/06/00  04:33p      <DIR>      .
```

```

                                test
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p      489 Entries
12/16/99  06:19p      66 Repository
12/16/99  06:19p      46 Root
12/16/99  06:19p      846 Template
                        6 File(s)      1,447 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\management

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:40p      372 Makefile
12/07/99  04:40p      16,612 ManagedAbstractResource.java
                        5 File(s)      16,984 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\management\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p      113 Entries
12/16/99  06:19p      69 Repository
12/16/99  06:19p      46 Root
12/16/99  06:19p      846 Template
                        6 File(s)      1,074 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\naming

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:40p      5,107 Binding.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:40p      454 Makefile
12/07/99  04:40p      23,229 MappingObject.java
12/07/99  04:40p      94,368 NameFrame.java
12/07/99  04:40p      8,209 NameSearchPolicyMatcher.java
12/07/99  04:40p      6,580 PartialBinding.java
                        9 File(s)      137,947 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\naming\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p      324 Entries

```

```

                                test
12/16/99  06:19p                65 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                6 File(s)          1,281 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\pd

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:40p                377 Makefile
12/07/99  04:40p            30,042 ProtectionDomain.java
12/07/99  04:40p            5,997 WorkingSet.java
                                6 File(s)          36,416 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\pd\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:19p                158 Entries
12/16/99  06:19p                61 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                6 File(s)          1,111 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\plugins

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:40p                393 Makefile
12/07/99  04:40p            3,495 PlugInLoader.java
12/07/99  04:40p            1,776 PlugInIF.java
12/07/99  04:40p            4,548 PlugIn.java
12/06/00  04:33p                <DIR>          secureboot
12/06/00  04:33p                <DIR>          testplugin
                                9 File(s)          10,212 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\plugins\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:19p                200 Entries
12/16/99  06:19p                40 Entries.Log

```

```

                                test
12/16/99  06:19p                66 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                7 File(s)
                                1,198 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\plugins\secureboot

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/06/00  04:33p                <DIR>      CVS
12/07/99  04:40p                4,395 SecureBoot.java
                                4 File(s)
                                4,395 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\plugins\secureboot\CVS

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/16/99  06:19p                55 Entries
12/16/99  06:19p                77 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                6 File(s)
                                1,024 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\plugins\testplugin

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/06/00  04:33p                <DIR>      CVS
12/07/99  04:41p                1,915 TestPlugIn.java
                                4 File(s)
                                1,915 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\plugins\testplugin\CVS

```

12/06/00  04:33p                <DIR>      .
12/06/00  04:33p                <DIR>      ..
12/16/99  06:19p                55 Entries
12/16/99  06:19p                77 Repository
12/16/99  06:19p                46 Root
12/16/99  06:19p                846 Template
                                6 File(s)
                                1,024 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\quota

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:41p                389 Makefile
12/07/99  04:41p            10,189 QuotaCheck.java
12/07/99  04:41p            14,951 QuotaSize.java
12/07/99  04:41p            25,082 Quota.java
                                7 File(s)
                                50,611 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\quota\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p            198 Entries
12/16/99  06:19p            64 Repository
12/16/99  06:19p            46 Root
12/16/99  06:19p            846 Template
                                6 File(s)
                                1,154 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\repository

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:41p            14,917 AbstractStore.java
12/07/99  04:41p            7,254 ActiveResourceOwners.java
12/07/99  04:41p            14,960 CacheEntry.java
12/07/99  04:41p            10,740 CacheStats.java
12/07/99  04:41p            30,996 Cache.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:41p            15,703 DumpCore.java
12/06/00  04:33p      <DIR>      jdbc
12/07/99  04:41p            771 Makefile
12/06/00  04:33p      <DIR>      mem
12/07/99  04:41p            17,870 RepositoryView.java
12/07/99  04:41p            4,437 RepositoryReset.java
12/07/99  04:41p            24,681 RepositoryHandle.java
12/07/99  04:41p            30,283 Repository.java
12/07/99  04:41p            29,593 Scavenger.java
12/07/99  04:41p            3,452 StoreUnitsIF.java
12/07/99  04:41p            8,091 StoreUnitList.java
12/07/99  04:41p            4,335 StoreUnit.java
                                20 File(s)
                                218,083 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\repository\CVS


```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:19p                      800 Entries
12/16/99  06:20p                      27 Entries.Log
12/16/99  06:19p                      69 Repository
12/16/99  06:19p                      46 Root
12/16/99  06:19p                      846 Template
                                7 File(s)          1,788 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\repository\jdbc

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:41p          14,197 JDBCRepositoryHandle.java
12/07/99  04:41p          29,056 JDBCHouseKeeping.java
12/07/99  04:41p          31,040 JDBCResourceState.java
12/07/99  04:41p          13,912 JDBCResourceSpecification.java
12/07/99  04:41p          10,243 JDBCResourceLookup.java
12/07/99  04:41p          22,361 JDBCResourceDescription.java
12/07/99  04:41p          17,317 JDBCSqlStrings.java
12/07/99  04:41p          16,847 JDBCStore.java
                                576 Makefile
                                12 File(s)        155,549 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\repository\jdbc\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:20p          526 Entries
12/16/99  06:19p          74 Repository
12/16/99  06:19p          46 Root
12/16/99  06:19p          846 Template
                                6 File(s)          1,492 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\repository\mem

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:41p          381 Makefile
12/07/99  04:41p          7,591 MemStoreDB.java
12/07/99  04:41p          14,777 MemStore.java
                                6 File(s)        22,749 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\repository\mem\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		150 Entries
12/16/99	06:20p		73 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
		6 File(s)	1,115 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\resource

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:41p		18,587 AbstractResource.java
12/07/99	04:41p		23,240 CoreManagementService.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:41p		15,970 ESUIDFactory.java
12/07/99	04:41p		6,218 ExternalResource.java
12/07/99	04:41p		576 Makefile
12/07/99	04:41p		59,946 MetaResource.java
12/07/99	04:41p		2,800 MutableResourceIF.java
12/07/99	04:41p		14,087 ResourceType.java
12/07/99	04:41p		12,611 ResourceSpecification.java
12/07/99	04:41p		17,749 ResourceFactory.java
		13 File(s)	171,784 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\resource\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		568 Entries
12/16/99	06:20p		67 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
		6 File(s)	1,527 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\router

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:41p		369 Makefile
12/07/99	04:41p		2,282 Response.java

12/07/99	04:41p	test	
		17,060 Router.java	
	6 File(s)	19,711 bytes	

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\router\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		146 Entries
12/16/99	06:20p		65 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
	6 File(s)		1,103 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\security

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:41p		1,824 KeyIF.java
12/07/99	04:41p		10,631 KeyRing.java
12/07/99	04:41p		7,550 Key.java
12/07/99	04:41p		456 Makefile
12/07/99	04:41p		24,731 MD5.java
12/07/99	04:41p		9,913 SecurityContextFactory.java
12/07/99	04:41p		3,095 SecurityContext.java
	10 File(s)		58,200 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\security\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		355 Entries
12/16/99	06:20p		67 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
	6 File(s)		1,314 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\startup

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:41p		12,653 CoreArgs.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:41p		414 Makefile

```

                                test
12/07/99  04:41p                2,166 pingServer.real
12/07/99  04:41p                5,724 Server.java
12/07/99  04:41p                6,003 StartESCore.java
12/07/99  04:41p               35,876 StartupCore.java
                                9 File(s)
                                62,836 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\startup\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:20p                304 Entries
12/16/99  06:20p                66 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p               846 Template
                                6 File(s)
                                1,262 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\sysmon

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:41p                357 Makefile
12/07/99  04:41p               8,817 SystemMonitor.java
                                5 File(s)
                                9,174 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\sysmon\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:20p               103 Entries
12/16/99  06:20p                65 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p               846 Template
                                6 File(s)
                                1,060 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\util

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/07/99  04:41p               24,650 CoreHelper.java
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:41p               35,464 Logger.java
12/07/99  04:41p                395 Makefile
12/07/99  04:41p               7,463 ServiceRegistry.java

```

test
7 File(s) 67,972 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\util\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		205 Entries
12/16/99	06:20p		63 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
6 File(s)			1,160 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\vocabulary

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:41p		24,331 Expression.java
12/07/99	04:41p		3,717 ExprNode.java
12/07/99	04:41p		9,836 LiteralNode.java
12/07/99	04:41p		480 Makefile
12/07/99	04:41p		19,310 OpNode.java
12/07/99	04:41p		5,454 PropertyNode.java
12/07/99	04:41p		17,130 Tokenizer.java
12/07/99	04:41p		16,969 Vocabulary.java
11 File(s)			97,227 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\core\vocabulary\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		408 Entries
12/16/99	06:20p		69 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
6 File(s)			1,369 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:19p		48 Entries
12/16/99	06:20p		78 Entries.Log
12/16/99	06:19p		53 Repository

```

12/16/99 06:19p      test
12/16/99 06:19p      46 Root
12/16/99 06:19p      846 Template
7 File(s)            1,071 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/06/00 04:33p      <DIR>      confactory
12/06/00 04:33p      <DIR>      connectors
12/06/00 04:33p      <DIR>      CVS
12/06/00 04:33p      <DIR>      esip
12/07/99 04:26p      303 Makefile
12/06/00 04:33p      <DIR>      proxy
12/06/00 04:33p      <DIR>      stacks
12/06/00 04:33p      <DIR>      util
10 File(s)          303 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\confactory

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p      7,252 CFEEventPublisher.java
12/07/99 04:26p      6,288 CFStatistics.java
12/06/00 04:33p      <DIR>      co
12/07/99 04:26p      2,138 CommunicationManagerException.j
ava
12/07/99 04:26p      65,437 CommunicationManager.java
12/07/99 04:26p      7,656 ConnectionRegistry.java
12/07/99 04:26p      3,671 ConnectionEntry.java
12/07/99 04:26p      3,309 ConnFactoryTags.java
12/07/99 04:26p      6,409 ConnFactorySM.java
12/07/99 04:26p      4,682 ConnFactoryInfo.java
12/07/99 04:26p      2,133 CreateProxyException.java
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      502 Makefile
12/06/00 04:33p      <DIR>      policy
12/07/99 04:26p      3,523 StatsUpdate.java
17 File(s)          113,000 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\confactory\co

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p      6,484 ConnectionObject.java

```

```

                                test
12/07/99  04:26p                                2,892 ConnectionInfoIntf.java
12/06/00  04:33p                                <DIR>                                CVS
12/07/99  04:26p                                2,515 SerializationException.java
12/07/99  04:26p                                5,225 TCPConnectionInfo.java
                                7 File(s)                                17,116 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\confactory\Co\CVS

```

12/06/00  04:33p                                <DIR>                                .
12/06/00  04:33p                                <DIR>                                ..
12/16/99  06:20p                                244 Entries
12/16/99  06:20p                                80 Repository
12/16/99  06:20p                                46 Root
12/16/99  06:20p                                846 Template
                                6 File(s)                                1,216 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\confactory\CVS

```

12/06/00  04:33p                                <DIR>                                .
12/06/00  04:33p                                <DIR>                                ..
12/16/99  06:20p                                694 Entries
12/16/99  06:20p                                28 Entries.Log
12/16/99  06:20p                                77 Repository
12/16/99  06:20p                                46 Root
12/16/99  06:20p                                846 Template
                                7 File(s)                                1,691 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\confactory\policy

```

12/06/00  04:33p                                <DIR>                                .
12/06/00  04:33p                                <DIR>                                ..
12/06/00  04:33p                                <DIR>                                CVS
12/07/99  04:26p                                13,446 Negotiator.java
12/07/99  04:26p                                14,105 NegotiationPolicy.java
12/07/99  04:26p                                4,684 NegotiationOffer.java
12/07/99  04:26p                                2,348 NegotiationFailedException.java
12/07/99  04:26p                                3,337 SecurityPolicy.java
                                8 File(s)                                37,920 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\confactory\policy\CVS

```

12/06/00  04:33p                                <DIR>                                .
12/06/00  04:33p                                <DIR>                                ..
12/16/99  06:20p                                296 Entries

```

```

                                test
12/16/99  06:20p                84 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                6 File(s)
                                1,272 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\connectors

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/07/99  04:26p                7,732 ConnectorManager.java
12/07/99  04:26p                6,183 ConnectorDesc.java
12/07/99  04:26p                4,495 Connector.java
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:26p                5,514 DataConnector.java
12/07/99  04:26p                3,714 ListeningConnector.java
12/07/99  04:26p                557 Makefile
12/07/99  04:26p                13,703 TcpDataConnector.java
12/07/99  04:26p                7,334 TcpListeningConnector.java
12/07/99  04:26p                6,420 TcpTunnelConnector.java
                                12 File(s)
                                55,652 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\connectors\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:20p                508 Entries
12/16/99  06:20p                77 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                6 File(s)
                                1,477 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:20p                48 Entries
12/16/99  06:20p                99 Entries.Log
12/16/99  06:20p                66 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                7 File(s)
                                1,105 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\esip


```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                      3,548 ESIPControlMessage.java
12/07/99  04:26p                      5,727 ESIPExportMessage.java
12/07/99  04:26p                      4,267 ESIPHeader.java
12/07/99  04:26p                      4,123 ESIPMessage.java
12/07/99  04:26p                      2,126 ESIPPayload.java
12/08/99  03:16p                      461 Makefile
                                9 File(s)      20,252 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\esip\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:20p                      325 Entries
12/16/99  06:20p                      71 Repository
12/16/99  06:20p                      46 Root
12/16/99  06:20p                      846 Template
                                6 File(s)      1,288 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\proxy

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                      2,166 EbyteArrayOutputStream.java
12/07/99  04:26p                      533 Makefile
12/07/99  04:26p                      12,689 ProxyAPIHandler.java
12/07/99  04:26p                      27,950 ProxyExportHandler.java
12/07/99  04:26p                      8,861 ProxyHelper.java
12/07/99  04:26p                      9,709 ProxyImportHandler.java
12/07/99  04:26p                      36,159 Proxy.java
12/07/99  04:26p                      26,195 StateMgr.java
12/07/99  04:26p                      3,882 Timeouts.java
                                12 File(s)      128,144 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\proxy\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:20p                      489 Entries
12/16/99  06:20p                      72 Repository
12/16/99  06:20p                      46 Root
12/16/99  06:20p                      846 Template

```

test
6 File(s) 1,453 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\stacks

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		5,443 LayerFactory.java
12/06/00	04:33p	<DIR>	layers
12/07/99	04:26p		457 Makefile
12/07/99	04:26p		4,429 StackBuilder.java
12/07/99	04:26p		14,775 Stack.java
		8 File(s)	25,104 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\stacks\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		203 Entries
12/16/99	06:20p		16 Entries.Log
12/16/99	06:20p		73 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
		7 File(s)	1,184 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\stacks\layers

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		2,166 CILayer.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		3,461 LayerIntf.java
12/07/99	04:26p		4,834 Layer.java
12/07/99	04:26p		464 Makefile
12/07/99	04:26p		7,071 NameAuthenticationLayer.java
12/07/99	04:26p		2,221 TransportLayer.java
		9 File(s)	20,217 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\stacks\layers\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		316 Entries
12/16/99	06:20p		80 Repository

```

12/16/99 06:20p      test
12/16/99 06:20p      46 Root
12/16/99 06:20p      846 Template
6 File(s)           1,288 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\util

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p      3,820 CondVar.java
12/07/99 04:26p      7,348 ConnectionObjectFileReader.java
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      4,748 Header.java
12/07/99 04:26p      3,180 Lock.java
12/07/99 04:26p      439 Makefile
12/07/99 04:26p      3,276 Synchronizer.java
12/07/99 04:26p      3,090 VerifyDomain.java
10 File(s)         25,901 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\intercorecom\util\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:20p      367 Entries
12/16/99 06:20p      71 Repository
12/16/99 06:20p      46 Root
12/16/99 06:20p      846 Template
6 File(s)           1,330 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      2,931 ESXMLDecoderIntf.java
12/07/99 04:26p      49,369 ESXMLDecoder.java
12/07/99 04:26p      2,453 ESXMLEncoderIntf.java
12/07/99 04:26p      6,183 ESXMLEncoder.java
12/07/99 04:26p      2,761 ESXMLShellIntf.java
12/07/99 04:26p      20,001 ESXMLShell.java
12/06/00 04:33p      <DIR>      IE5
12/07/99 04:26p      515 Makefile
12/06/00 04:33p      <DIR>      schemas
12/07/99 04:26p      7,355 XMLApp.java
12/07/99 04:26p      4,425 XMLNameList.java
12/06/00 04:33p      <DIR>      xmlserver

```

15 File(s) test
95,993 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		481 Entries
12/16/99	06:20p		49 Entries.Log
12/16/99	06:20p		57 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
7 File(s)			1,479 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\IE5

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		854 e-speak_logo.gif
12/07/99	04:26p		1,792 lookup.htm
12/07/99	04:26p		3,320 lookup.js
12/07/99	04:26p		628 Readme.txt
12/07/99	04:26p		1,623 register.htm
12/07/99	04:26p		3,226 register.js
12/07/99	04:26p		3,631 response.js
12/07/99	04:26p		1,757 vocab.htm
12/07/99	04:26p		3,873 vocab.js
12 File(s)			20,704 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\IE5\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:20p		432 Entries
12/16/99	06:20p		61 Repository
12/16/99	06:20p		46 Root
12/16/99	06:20p		846 Template
6 File(s)			1,385 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\schemas

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS

```

                                test
12/07/99  04:26p                4,405 es-basic.xsd
12/07/99  04:26p                368 es-lookup.xsd
12/07/99  04:26p                419 es-register.xsd
12/07/99  04:26p                589 es-response.xsd
12/07/99  04:26p                498 es-vocab.xsd
                                8 File(s)        6,279 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\schemas\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:20p                255 Entries
12/16/99  06:20p                65 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                6 File(s)        1,212 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\xmlserver

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:26p                5,291 ESXMLHttp.java
12/07/99  04:26p                3,218 ESXMLServerThread.java
12/07/99  04:26p                3,397 ESXMLServer.java
12/07/99  04:26p                314 Makefile
12/07/99  04:26p                8,305 TestServerApp.java
                                8 File(s)        20,525 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\infra\xml\xmlserver\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:20p                266 Entries
12/16/99  06:20p                67 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                6 File(s)        1,225 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/07/99  04:26p                7,741 AppSessionIDFilter.java

```

12/07/99	04:26p		test
12/07/99	04:26p		9,404 ClassDependencyFinder.java
12/06/00	04:33p	<DIR>	9,098 ClassFinder.java
12/07/99	04:26p		CVS
12/07/99	04:26p		3,496 ESContract.java
12/07/99	04:26p		55,605 ESAbstractFinder.java
12/07/99	04:26p		38,691 ESAbstractElement.java
12/07/99	04:26p		8,783 ESAccessRight.java
12/07/99	04:26p		47,370 ESAccessorStub.java
12/07/99	04:26p		2,293 ESAccessorHandle.java
12/07/99	04:26p		17,372 ESAccessor.java
12/07/99	04:26p		4,455 ESAppSessionID.java
12/07/99	04:26p		3,690 ESAssert.java
12/07/99	04:26p		9,610 ESAttribute.java
12/07/99	04:26p		17,621 ESBaseServiceStub.java
12/07/99	04:26p		2,226 ESBaseService.java
12/07/99	04:26p		9,288 ESBaseQuery.java
12/07/99	04:26p		22,360 ESBaseDescription.java
12/07/99	04:26p		7,475 ESServiceDescription.java
12/07/99	04:26p		5,151 ESVocabularyElement.java
12/07/99	04:26p		2,374 ESService.java
12/07/99	04:26p		4,939 ESCallbackIntf.java
12/07/99	04:26p		7,132 ESCallbackImpl.java
12/07/99	04:26p		2,478 ESCloneable.java
12/07/99	04:26p		12,068 ESCommunity.java
12/07/99	04:26p		11,751 ESConfiguration.java
12/07/99	04:26p		18,190 ESConnector.java
12/07/99	04:26p		47,841 ESConnection.java
12/07/99	04:26p		6,580 ESConstants.java
12/07/99	04:26p		8,475 ESContractStub.java
12/07/99	04:26p		6,541 ESContractFinder.java
12/07/99	04:26p		7,474 ESContractElement.java
12/07/99	04:26p		7,914 ESContractDescription.java
12/07/99	04:26p		44,996 ESServiceElement.java
12/07/99	04:26p		12,517 ESServiceContext.java
12/07/99	04:26p		6,168 ESFolderFinder.java
12/07/99	04:26p		55,878 ESFolder.java
12/07/99	04:26p		3,806 ESHashMap.java
12/07/99	04:26p		6,331 ESHelper.java
12/07/99	04:26p		5,295 ESImplProxy.java
12/07/99	04:26p		2,228 ESIntrospectionIntf.java
12/07/99	04:26p		5,026 ESIntrospectionImpl.java
12/07/99	04:26p		9,790 ESList.java
12/07/99	04:26p		1,977 ESManagementIntf.java
12/07/99	04:26p		2,633 ESMessageRegistryIntf.java
12/07/99	04:26p		4,917 ESMessage.java
12/07/99	04:26p		14,917 ESMPM.java
12/07/99	04:26p		2,388 ESNameFactory.java
12/07/99	04:26p		3,755 ESQuery.java

```

test
12/07/99 04:26p      3,189 ESRunnable.java
12/07/99 04:26p    13,633 ESScope.java
12/07/99 04:26p     2,564 ESSecurityIntf.java
12/07/99 04:26p     8,649 ESServiceStub.java
12/07/99 04:26p    18,986 ESServiceMessenger.java
12/07/99 04:26p    21,090 ESServiceHandler.java
12/07/99 04:26p     8,170 ESServiceFinder.java
12/07/99 04:26p     2,507 ESSessionIntf.java
12/07/99 04:26p     4,693 ESSessionImpl.java
12/07/99 04:26p    18,154 ESStartService.java
12/07/99 04:26p     2,223 ESStubFactoryIntf.java
12/07/99 04:26p     6,867 ESStubFactoryImpl.java
12/07/99 04:26p     4,213 ESThread.java
12/07/99 04:26p     4,909 ESUniqList.java
12/07/99 04:26p     6,042 ESUserCredential.java
12/07/99 04:26p    20,618 ESVocabularyDescription.java
12/07/99 04:26p     2,868 ESVocabulary.java
12/07/99 04:26p     8,682 ESVocabularyStub.java
12/07/99 04:26p     6,647 ESVocabularyFinder.java
12/07/99 04:26p     3,243 ESXMLFile.java
12/07/99 04:26p     5,276 ESXMLQuery.java
12/06/00 04:33p      <DIR>      event
12/07/99 04:26p      5,290 GenericClassLoader.java
12/07/99 04:26p      2,169 Makefile
12/06/00 04:33p      <DIR>      management
                        76 File(s)      788,790 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je
si\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:20p      3,944 Entries
12/16/99 06:20p        35 Entries.Log
12/16/99 06:20p        52 Repository
12/16/99 06:20p        46 Root
12/16/99 06:20p        846 Template
                        7 File(s)      4,923 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je
si\event

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/06/00 04:33p      <DIR>      coredist
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p    15,259 ESCoreSubscriber.java
12/07/99 04:26p     2,359 ESCoreListenerIntf.java

```

```

test
12/07/99 04:26p 14,426 ESDistributor.java
12/07/99 04:26p 6,481 ESListenerStub.java
12/07/99 04:26p 2,777 ESListenerIntf.java
12/07/99 04:26p 5,660 ESPublisherStub.java
12/07/99 04:26p 2,589 ESPublisherIntf.java
12/07/99 04:26p 20,502 ESPublisher.java
12/07/99 04:26p 17,999 ESSubscriber.java
12/07/99 04:26p 2,059 EventException.java
12/06/00 04:33p <DIR> impl
12/06/00 04:33p <DIR> intf
12/07/99 04:26p 730 Makefile
12/07/99 04:26p 5,740 PublisherInfo.java
12/07/99 04:26p 4,843 ResultSetEntry.java
12/07/99 04:26p 5,769 ResultSet.java
12/06/00 04:33p <DIR> util
21 File(s) 107,193 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi\event\coredist

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 6,908 CoreSubProfile.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 3,621 ESCoreListenerStub.java
12/07/99 04:26p 4,352 ESCoreDistributorStub.java
12/07/99 04:26p 3,343 ESCoreDistributorIntf.java
12/07/99 04:26p 8,886 ESCoreDistributorImpl.java
12/07/99 04:26p 485 Makefile
9 File(s) 27,595 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi\event\coredist\CVS

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:20p 353 Entries
12/16/99 06:20p 67 Repository
12/16/99 06:20p 46 Root
12/16/99 06:20p 846 Template
6 File(s) 1,312 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi\event\CVS

```

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:20p 772 Entries

```



```

                                test
12/16/99  06:20p                60 Entries.Log
12/16/99  06:20p                58 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                7 File(s)          1,782 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi\event\impl

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:26p                11,967 DistributorStore.java
12/07/99  04:26p                10,316 ESEventServiceComponentHelper.j
ava
12/07/99  04:26p                17,224 ESEventServiceComponent.java
12/07/99  04:26p                12,805 ESEventDistributorImpl.java
12/07/99  04:26p                510 Makefile
12/07/99  04:26p                5,724 RegisterEventABI.java
12/07/99  04:26p                4,909 SubsDet.java
                                10 File(s)        63,455 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi\event\impl\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:20p                413 Entries
12/16/99  06:20p                63 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                6 File(s)          1,368 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\jesi\event\intf

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:26p                10,752 ESDistributorStub.java
12/07/99  04:26p                3,317 ESDistributorIntfMessageRegistr
y.java
12/07/99  04:26p                4,324 ESDistributorIntf.java
12/07/99  04:26p                11,792 ESEventServiceStub.java
12/07/99  04:26p                3,320 ESEventServiceIntfMessageRegistr
y.java
12/07/99  04:26p                2,865 ESEventServiceIntf.java
12/07/99  04:26p                4,945 ESListenerIntfMessageRegistry.j

```

test

```
ava
12/07/99 04:26p          3,346 ESPublisherIntfMessageRegistry.
java
12/07/99 04:26p          613 Makefile
          12 File(s)      45,274 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je
si\event\intf\CVS

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:20p          578 Entries
12/16/99 06:20p          63 Repository
12/16/99 06:20p          46 Root
12/16/99 06:20p          846 Template
          6 File(s)      1,533 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je
si\event\util

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p          4,686 CoreEventPredicate.java
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p          5,530 EventContext.java
12/07/99 04:26p          3,783 EventDataTypes.java
12/07/99 04:26p          8,788 EventMapEntry.java
12/07/99 04:26p          8,550 EventMap.java
12/07/99 04:26p          8,253 EventPredicate.java
12/07/99 04:26p          5,526 Id.java
12/07/99 04:26p          6,323 InputInfo.java
12/07/99 04:26p          551 Makefile
12/07/99 04:26p          6,391 PubProfile.java
12/07/99 04:26p          9,576 SubProfile.java
          14 File(s)      67,957 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je
si\event\util\CVS

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:20p          578 Entries
12/16/99 06:20p          63 Repository
12/16/99 06:20p          46 Root
12/16/99 06:20p          846 Template
          6 File(s)      1,533 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je

test

si\management

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p      12,092 AbstractManagedService.java
12/07/99  04:26p      2,703 CoreConnectionException.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p      2,672 IllegalStateTransition.esidl
12/07/99  04:26p      2,415 IllegalStateTransition.java
12/07/99  04:26p      1,135 Makefile
12/07/99  04:26p      12,899 ManagedServiceIntf.esidl
12/07/99  04:26p      15,798 ManagedServiceStub.java
12/07/99  04:26p      2,306 ManagedServiceIntfMessageRegist
ry.java
12/07/99  04:26p      12,899 ManagedServiceIntf.java
12/07/99  04:26p      3,496 ManageableService.java
12/07/99  04:26p      2,680 NoSuchVariableName.esidl
12/07/99  04:26p      2,395 NoSuchVariableName.java
12/07/99  04:26p      1,748 ResourceEntry.esidl
12/07/99  04:26p      2,800 ResourceEntry.java
12/07/99  04:26p      6,580 ServiceContext.java
12/07/99  04:26p      6,096 SimpleManagedService.java
12/07/99  04:26p      1,770 VariableType.esidl
12/07/99  04:26p      3,097 VariableType.java
12/07/99  04:26p      1,965 VariableEntry.esidl
12/07/99  04:26p      2,734 VariableEntry.java
                23 File(s)      100,280 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\je
si\management\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:20p      1,191 Entries
12/16/99  06:20p      63 Repository
12/16/99  06:20p      46 Root
12/16/99  06:20p      846 Template
                6 File(s)      2,146 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      advertise
12/07/99  04:26p      149 compile.bat
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      events
```

```

                                test
12/07/99  04:26p                                269 Makefile
12/06/00  04:33p                                management
12/06/00  04:33p                                <DIR> tunnel
                                9 File(s)          418 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise

```

12/06/00  04:33p                                <DIR> .
12/06/00  04:33p                                <DIR> ..
12/06/00  04:33p                                <DIR> agents
12/06/00  04:33p                                <DIR> bin
12/06/00  04:33p                                <DIR> clientapi
12/07/99  04:26p                                1,055 compile.bat
12/07/99  04:26p                                143 co.MYCOs
12/06/00  04:33p                                <DIR> CVS
12/06/00  04:33p                                <DIR> discovery
12/06/00  04:33p                                <DIR> exception
12/07/99  04:26p                                325 Makefile
12/07/99  04:26p                                668 makefile.release
12/07/99  04:26p                                131 object-class-definition
12/06/00  04:33p                                <DIR> query_parser
12/06/00  04:33p                                <DIR> util
12/06/00  04:33p                                <DIR> ypcommon
12/06/00  04:33p                                <DIR> ypserver
                                17 File(s)          2,322 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\agents

```

12/06/00  04:33p                                <DIR> .
12/06/00  04:33p                                <DIR> ..
12/07/99  04:26p                                14,434 AdvLSAgent.java
12/07/99  04:26p                                5,019 BackendAgentIntf.java
12/06/00  04:33p                                <DIR> CVS
12/07/99  04:26p                                45,917 LDAPDirAgent.java
12/07/99  04:26p                                441 Makefile
12/07/99  04:26p                                6,372 ProxySocketFactory.java
                                8 File(s)          72,183 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\agents\CVS

```

12/06/00  04:33p                                <DIR> .
12/06/00  04:33p                                <DIR> ..
12/16/99  06:20p                                272 Entries
12/16/99  06:20p                                73 Repository
12/16/99  06:20p                                46 Root

```

12/16/99 06:20p test
846 Template
6 File(s) 1,237 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\bin

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 607 runas_ldap.bat
12/07/99 04:26p 640 runas_ldap
12/07/99 04:26p 599 runas_noldap
12/07/99 04:26p 580 runas_noldap.bat
12/07/99 04:26p 585 runas_ldap_s
12/07/99 04:26p 549 runas_ldap_s.bat
12/07/99 04:26p 585 runas_ldap_i_s
12/07/99 04:26p 549 runas_ldap_i_s.bat
12/07/99 04:26p 673 runas_ldap_i
12/07/99 04:26p 635 runas_ldap_i.bat
13 File(s) 6,002 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\bin\CVS

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:20p 513 Entries
12/16/99 06:20p 70 Repository
12/16/99 06:20p 46 Root
12/16/99 06:20p 846 Template
6 File(s) 1,475 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\clientapi

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 23,009 ESADServiceInterface.java
12/07/99 04:26p 376 Makefile
5 File(s) 23,385 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\clientapi\CVS

12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:21p 110 Entries

```

                                test
12/16/99  06:20p                76 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                6 File(s)          1,078 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:20p                254 Entries
12/16/99  06:21p                158 Entries.Log
12/16/99  06:20p                66 Repository
12/16/99  06:20p                46 Root
12/16/99  06:20p                846 Template
                                7 File(s)          1,370 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\discovery

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/07/99  04:26p                4,005 AdvDscvMsgBag.java
12/07/99  04:26p                1,992 AdvDscvMsg.java
12/07/99  04:26p                4,022 AdvIncomingDscvRqst.java
12/07/99  04:26p                4,575 AdvIncomingDscvRply.java
12/07/99  04:26p            12,183 AdvMcastSenderReceiver.java
12/07/99  04:26p                3,810 AdvOutgoingDscvRqst.java
12/07/99  04:26p                7,247 AdvOutgoingDscvRply.java
12/07/99  04:26p                4,524 AdvSLPOutgoingMsg.java
12/07/99  04:26p                3,336 AdvSLPMsg.java
12/07/99  04:26p                7,123 AdvSLPIncomingMsg.java
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:26p                640 Makefile
                                14 File(s)          53,457 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\discovery\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:21p                632 Entries
12/16/99  06:21p                76 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                6 File(s)          1,600 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se

rvices\advertise\exception test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 3,118 AdvException.java
12/07/99 04:26p 3,237 AdvNoSuchBackendException.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 3,089 IllegalArgument.java
12/07/99 04:26p 490 Makefile
12/07/99 04:26p 3,175 NoADServiceException.java
12/07/99 04:26p 3,148 ParseException.java
          9 File(s) 16,257 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\advertise\exception\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:21p 349 Entries
12/16/99 06:21p 76 Repository
12/16/99 06:21p 46 Root
12/16/99 06:21p 846 Template
          6 File(s) 1,317 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\advertise\query_parser

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 2,505 LDAPQueryOps.java
12/07/99 04:26p 747 Makefile
12/07/99 04:26p 3,408 QueryLeafNode.java
12/07/99 04:26p 4,285 QueryAndNode.java
12/07/99 04:26p 5,451 QueryComparatorNode.java
12/07/99 04:26p 4,077 QueryExprNode.java
12/07/99 04:26p 3,667 QueryEqNode.java
12/07/99 04:26p 3,095 QueryGtNode.java
12/07/99 04:26p 3,101 QueryGeNode.java
12/07/99 04:26p 2,252 QueryIdentifierNode.java
12/07/99 04:26p 3,073 QueryLtNode.java
12/07/99 04:26p 2,636 QueryLiteralNode.java
12/07/99 04:26p 17,430 QueryLex.java
12/07/99 04:26p 3,097 QueryLeNode.java
12/07/99 04:26p 3,498 QueryNotNode.java
12/07/99 04:26p 3,825 QueryNeqNode.java
12/07/99 04:26p 4,373 QueryOrNode.java
12/07/99 04:26p 14,678 QueryParser.java
```

21 File(s) test
85,198 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\query_parser\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		975 Entries
12/16/99	06:21p		79 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
6 File(s)			1,946 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\util

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		3,428 eskill.java
12/07/99	04:26p		424 Makefile
12/07/99	04:26p		5,560 Multicaster.java
12/07/99	04:26p		2,191 ShutdownIntf.java
12/07/99	04:26p		6,600 ShutdownCtrl.java
8 File(s)			18,203 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\util\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		257 Entries
12/16/99	06:21p		71 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
6 File(s)			1,220 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\ypcommon

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		2,249 AdvDebug.java
12/07/99	04:26p		6,157 ArraySet.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		689 Makefile
12/07/99	04:26p		2,465 QueryOps.java
12/07/99	04:26p		2,537 YPCompBigDecimal.java

12/07/99	04:26p	test
12/07/99	04:26p	2,451 YPCompFloat.java
12/07/99	04:26p	9,801 YPComparator.java
12/07/99	04:26p	2,430 YPCompByte.java
12/07/99	04:26p	2,442 YPCompDouble.java
12/07/99	04:26p	2,566 YPComparable.java
12/07/99	04:26p	2,353 YPCompString.java
12/07/99	04:26p	2,430 YPCompShort.java
12/07/99	04:26p	2,418 YPCompLong.java
12/07/99	04:26p	2,439 YPCompInt.java
12/07/99	04:26p	2,504 YPCompChar.java
12/07/99	04:26p	2,087 YPObjects.java
12/07/99	04:26p	2,745 YPOperations.java
	20 File(s)	50,763 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\ypcommon\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		890 Entries
12/16/99	06:21p		75 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
	6 File(s)		1,857 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\ypserver

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		88,292 AdvertisingService.java
12/07/99	04:26p		4,168 BackendLookupResult.java
12/07/99	04:26p		5,140 BackendConnectionInfo.java
12/07/99	04:26p		16,210 ConnectionManager.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		4,733 ExternalEsrl.java
12/07/99	04:26p		651 Makefile
12/07/99	04:26p		4,138 TimeoutHandler.java
12/07/99	04:26p		5,346 YPGenericTable.java
12/07/99	04:26p		5,438 YPItemsTable.java
12/07/99	04:26p		4,980 YPItem.java
12/07/99	04:26p		3,221 YPMessage.java
12/07/99	04:26p		3,821 YPSearchResultsTable.java
12/07/99	04:26p		9,852 YPSearchResult.java
	16 File(s)		155,990 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\advertise\ypserver\CVS

test

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:21p                728 Entries
12/16/99 06:21p                75 Repository
12/16/99 06:21p                46 Root
12/16/99 06:21p                846 Template
      6 File(s)                1,695 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\CVS

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:20p                96 Entries
12/16/99 06:21p                71 Entries.Log
12/16/99 06:20p                56 Repository
12/16/99 06:20p                46 Root
12/16/99 06:20p                846 Template
      7 File(s)                1,115 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\events

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p                697 compile.bat
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p                5,926 ESCoreDistributor.java
12/07/99 04:26p                5,093 ESManagementDistributor.java
12/07/99 04:26p                3,995 ESServiceDistributor.java
12/07/99 04:26p                465 Makefile
12/07/99 04:26p                510 makefile.release
      9 File(s)                16,686 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\events\CVS

```
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:21p                335 Entries
12/16/99 06:21p                63 Repository
12/16/99 06:21p                46 Root
12/16/99 06:21p                846 Template
      6 File(s)                1,290 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 1,328 compile.bat
12/06/00 04:33p <DIR> coremanager
12/06/00 04:33p <DIR> CVS
12/06/00 04:33p <DIR> logger
12/07/99 04:26p 333 Makefile
12/07/99 04:26p 759 makefile.release
12/06/00 04:33p <DIR> policymanager
12/06/00 04:33p <DIR> processmanager
12/06/00 04:33p <DIR> servicemanager
12/06/00 04:33p <DIR> web
12 File(s) 2,420 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\coremanager

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 27,653 CoreManager.java
12/07/99 04:26p 20,917 CoreManagerMSImpl.java
12/07/99 04:26p 12,661 CoreManagerOS.java
12/07/99 04:26p 19,588 CoreManagerMS.java
12/07/99 04:26p 705 compile.bat
12/07/99 04:26p 7,599 CoreEventSubscriberImpl.java
12/07/99 04:26p 5,247 CoreEventSubscriber.java
12/07/99 04:26p 7,420 CoreManagerTags.java
12/07/99 04:26p 6,473 CoreManagerStats.java
12/07/99 04:26p 4,484 CoreManagerOSIntf.java
12/07/99 04:26p 23,110 CoreManagerOSImpl.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 4,148 DbgLog.java
12/07/99 04:26p 6,249 EventPublisher.java
12/07/99 04:26p 8,116 ExtEventSubscriberImpl.java
12/07/99 04:26p 4,622 ExtEventSubscriber.java
12/15/99 04:00p 931 Makefile
12/07/99 04:26p 527 makefile.release
12/07/99 04:26p 5,151 ServiceDataList.java
12/07/99 04:26p 3,528 Shutdown.java
12/07/99 04:26p 5,112 StateMC.java
12/07/99 04:26p 4,434 StaticInfo.java
12/07/99 04:26p 10,091 StatsReader.java
12/07/99 04:26p 3,785 StatsStore.java
12/07/99 04:26p 6,651 UpdateStats.java
27 File(s) 199,202 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se

test
rvices\management\coremanager\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		1,324 Entries
12/16/99	06:21p		79 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
		6 File(s)	2,295 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		149 Entries
12/16/99	06:21p		121 Entries.Log
12/16/99	06:21p		67 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
		7 File(s)	1,229 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\logger

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	client
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		333 Makefile
12/06/00	04:33p	<DIR>	manager
12/06/00	04:33p	<DIR>	message
12/06/00	04:33p	<DIR>	service
12/06/00	04:33p	<DIR>	test
12/06/00	04:33p	<DIR>	types
12/06/00	04:33p	<DIR>	util
		11 File(s)	333 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\logger\client

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		4,325 ESLogClient.java
12/15/99	04:00p		434 Makefile
		5 File(s)	4,759 bytes

test

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\client\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                      101 Entries
12/16/99  06:21p                      81 Repository
12/16/99  06:21p                      46 Root
12/16/99  06:21p                      846 Template
        6 File(s)                1,074 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                      48 Entries
12/16/99  06:21p                    110 Entries.Log
12/16/99  06:21p                      74 Repository
12/16/99  06:21p                      46 Root
12/16/99  06:21p                      846 Template
        7 File(s)                1,124 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\manager

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                    9,765 ESLogServiceServlet.java
12/07/99  04:26p                      69 ESLogServiceManager.properties
12/07/99  04:26p                    7,964 ESLogServiceManager.java
12/15/99  04:00p                      473 Makefile
        7 File(s)                18,271 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\manager\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                    237 Entries
12/16/99  06:21p                      82 Repository
12/16/99  06:21p                      46 Root
12/16/99  06:21p                      846 Template
        6 File(s)                1,211 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\message

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 13,666 ESLogMessage.java
12/07/99 04:26p 405 Makefile
5 File(s) 14,071 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\message\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:21p 102 Entries
12/16/99 06:21p 82 Repository
12/16/99 06:21p 46 Root
12/16/99 06:21p 846 Template
6 File(s) 1,076 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\service

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 18,657 ESLogFileWriter.java
12/07/99 04:26p 3,693 ESLoggerWaitQueue.java
12/07/99 04:26p 16,773 ESLogMessageParser.java
12/07/99 04:26p 2,281 ESLogServiceIntf.esidl
12/07/99 04:26p 4,131 ESLogServiceStub.java
12/07/99 04:26p 1,135 ESLogServiceIntfMessageRegistry
java
12/07/99 04:26p 2,281 ESLogServiceIntf.java
12/07/99 04:26p 3,737 ESLogServiceImpl.java
12/07/99 04:26p 741 Makefile
12/07/99 04:26p 2,727 ServicePackageWhiteBoxTest.java
12/07/99 04:26p 4,995 TestDenialOfService.java
12/07/99 04:26p 4,380 TestEscapeProcessing.java
12/07/99 04:26p 12,892 TestFileWriting.java
12/07/99 04:26p 6,023 TestQueueProcessing.java
17 File(s) 84,446 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\service\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:21p 839 Entries
```

```

                                test
12/16/99  06:21p                82 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                6 File(s)          1,813 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\test

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                2,058 ESLoggerTest.java
12/07/99  04:26p                4,297 ESLoggerDefaultTest.java
12/07/99  04:26p                403 Makefile
                                6 File(s)          6,758 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\test\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                163 Entries
12/16/99  06:21p                79 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                6 File(s)          1,134 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\types

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                3,888 ESLoggableDataItem.java
12/07/99  04:26p                4,928 ESLoggableDataItemList.java
12/07/99  04:26p                3,485 ESLoggableBoolean.java
12/07/99  04:26p                3,481 ESLoggableString.java
12/07/99  04:26p                3,531 ESLoggableInteger.java
12/07/99  04:26p                3,529 ESLoggableFloat.java
12/07/99  04:26p                4,459 ESLoggableDate.java
12/07/99  04:26p                547 Makefile
                                11 File(s)         27,848 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\types\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..

```

```

                                test
12/16/99  06:21p                461 Entries
12/16/99  06:21p                80 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                6 File(s)      1,433 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\util

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                2,286 ESLoggableDateFormat.java
12/07/99  04:26p                381 Makefile
                                5 File(s)      2,667 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\logger\util\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                110 Entries
12/16/99  06:21p                79 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                6 File(s)      1,081 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\policymanager

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      filingsystem
12/15/99  04:00p                1,397 Makefile
12/07/99  04:26p                2,330 PersistenceFailureException.java
12/07/99  04:26p                2,207 PolicyLifecycleListener.java
12/07/99  04:26p                5,198 PolicyManagerFactoryIntf.java
12/07/99  04:26p                30,027 PolicyManagerFactoryImpl.java
12/07/99  04:26p                2,145 PolicyManagerFactoryException.java
12/07/99  04:26p                3,828 PolicyLifecycleEvent.java
12/07/99  04:26p                4,792 PolicyManagerIntf.java
12/07/99  04:26p                2,819 PolicyManagerLinkageEvent.java
12/07/99  04:26p                2,708 PolicyManagerLoopException.java
12/07/99  04:26p                1,817 PolicyManagerFactoryIntfMessage
Registry.java

```



```

12/07/99 04:26p      test
12/07/99 04:26p      6,568 PolicyManagerStub.java
12/07/99 04:26p      2,392 PolicyChangedEvent.java
12/07/99 04:26p      3,000 PolicyManagerLifecycleEvent.jav
a
12/07/99 04:26p      2,400 PolicyNotFoundException.java
12/07/99 04:26p      2,694 PolicyManagerNotLinkedException
.java
12/07/99 04:26p      2,247 PolicyManagerLifecycleListener.
java
12/07/99 04:26p      2,225 PolicyManagerLinkageListener.ja
va
12/07/99 04:26p      2,219 PolicyChangedListener.java
12/07/99 04:26p      2,549 PolicyManagerExistenceException
.java
12/07/99 04:26p      1,394 PolicyManagerIntfMessageRegistr
y.java
12/07/99 04:26p      5,720 PolicyManagerFactoryIntf.esidl
12/07/99 04:26p      24,384 PolicyManagerImpl.java
12/07/99 04:26p      4,792 PolicyManagerIntf.esidl
12/07/99 04:26p      6,112 PolicyManagerFactoryStub.java
12/07/99 04:26p      2,188 PolicyManagerFactoryShutdownLis
tener.java
12/07/99 04:26p      2,069 PolicyManagerFactoryShutdownEve
nt.java
12/06/00 04:33p      <DIR>      servlet
32 File(s)          132,221 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\policymanager\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:21p      1,805 Entries
12/16/99 06:21p      39 Entries.Log
12/16/99 06:21p      81 Repository
12/16/99 06:21p      46 Root
12/16/99 06:21p      846 Template
7 File(s)          2,817 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\policymanager\filingsystem

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p      8,047 ChangeList.java
12/07/99 04:26p      1,931 CorruptedPolicyFileException.ja
va
12/06/00 04:33p      <DIR>      CVS

```

```

                                test
12/07/99  04:26p                13,747 FileFormatter.java
12/07/99  04:26p                9,032 FileLoader.java
12/07/99  04:26p                5,168 FileUtil.java
12/07/99  04:26p               17,791 FilingSystemPersistencyManager.
java
12/07/99  04:26p                33 FilingSystemPersistencyManager.
properties
12/15/99  04:00p                590 Makefile
12/07/99  04:26p               11,933 PolicyManagerFile.java
                                68,272 bytes
                                12 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\policymanager\filingsystem\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                536 Entries
12/16/99  06:21p                94 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p               846 Template
                                6 File(s)
                                1,522 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\policymanager\servlet

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/15/99  04:00p                424 Makefile
12/07/99  04:26p               16,601 PolicyManagerServlet.java
                                5 File(s)
                                17,025 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\policymanager\servlet\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                110 Entries
12/16/99  06:21p                89 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p               846 Template
                                6 File(s)
                                1,091 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\processmanager

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..

```

```

                                test
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p      349 Makefile
12/06/00  04:33p      <DIR>      service
12/06/00  04:33p      <DIR>      shell
12/06/00  04:33p      <DIR>      statemonitor
12/06/00  04:33p      <DIR>      test
12/06/00  04:33p      <DIR>      util
12/06/00  04:33p      <DIR>      web
                                10 File(s)
                                349 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p      48 Entries
12/16/99  06:21p      95 Entries.Log
12/16/99  06:21p      82 Repository
12/16/99  06:21p      46 Root
12/16/99  06:21p      846 Template
                                7 File(s)
                                1,117 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\service

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p      652 Makefile
12/07/99  04:26p      2,226 ProcessManagerException.java
12/07/99  04:26p      4,494 ProcessManager.java
12/07/99  04:26p      5,155 ProcessManagerServiceIntf.esidl
12/07/99  04:26p      8,380 ProcessManagerServiceStub.java
12/07/99  04:26p      1,923 ProcessManagerException.esidl
12/07/99  04:26p      1,393 ProcessManagerServiceIntfMessage
Registry.java
12/07/99  04:26p      5,155 ProcessManagerServiceIntf.java
12/07/99  04:26p      10,069 ProcessManagerService.java
                                12 File(s)
                                39,447 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\service\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p      582 Entries
12/16/99  06:21p      90 Repository
12/16/99  06:21p      46 Root

```

```

                                test
12/16/99  06:21p                846 Template
                                1,564 bytes
        6 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\shell

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:26p                427 Makefile
12/07/99  04:26p                11,963 PMShellService.java
12/07/99  04:26p                24,700 PMShellConsole.java
12/07/99  04:26p                4,103 PMShell.java
        7 File(s)                41,193 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\shell\CVS

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/16/99  06:21p                209 Entries
12/16/99  06:21p                88 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
        6 File(s)                1,189 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\statemonitor

```

12/06/00  04:33p                <DIR>                .
12/06/00  04:33p                <DIR>                ..
12/07/99  04:26p                4,182 CoreConnectionMonitor.java
12/06/00  04:33p                <DIR>                CVS
12/07/99  04:26p                3,129 ESJobIntf.java
12/07/99  04:26p                1,868 ESJob.java
12/07/99  04:26p                3,731 InJVMJob.java
12/07/99  04:26p                641 Makefile
12/07/99  04:26p                4,007 MonitorIniFile.java
12/07/99  04:26p                5,260 ProcessJob.java
12/07/99  04:26p                3,947 StateMonitorIntf.java
12/07/99  04:26p                1,794 StateMonitorException.java
12/07/99  04:26p                4,679 StateMonitorConfig.java
12/07/99  04:26p                3,468 StateMonitor.java
12/07/99  04:26p                5,362 TaskErrStreamMonitor.java
        15 File(s)               42,068 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\statemonitor\CVS

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:21p 664 Entries
12/16/99 06:21p 95 Repository
12/16/99 06:21p 46 Root
12/16/99 06:21p 846 Template
6 File(s) 1,651 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 378 Makefile
12/07/99 04:26p 176 TEST.ESH
12/07/99 04:26p 2,416 TestTask.java
6 File(s) 2,970 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\test\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:21p 143 Entries
12/16/99 06:21p 87 Repository
12/16/99 06:21p 46 Root
12/16/99 06:21p 846 Template
6 File(s) 1,122 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\util

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:26p 3,136 ClientHelper.java
12/07/99 04:26p 3,815 CommandLine.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:26p 3,272 InJVMRunner.java
12/07/99 04:26p 2,979 IOPipe.java
12/07/99 04:26p 493 Makefile
12/07/99 04:26p 3,715 SimpleLoader.java
9 File(s) 17,410 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\util\CVS

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                      310 Entries
12/16/99  06:21p                      87 Repository
12/16/99  06:21p                      46 Root
12/16/99  06:21p                      846 Template
                                6 File(s)          1,289 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\web

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/15/99  04:00p                      447 Makefile
12/07/99  04:26p                  12,009 PMShellServlet.java
12/07/99  04:26p                      129 servlet.properties
                                6 File(s)          12,585 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\processmanager\web\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                      159 Entries
12/16/99  06:21p                      86 Repository
12/16/99  06:21p                      46 Root
12/16/99  06:21p                      846 Template
                                6 File(s)          1,137 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\servicemanager

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                      848 Makefile
12/07/99  04:26p                  9,820 ServiceManager.java
12/07/99  04:26p                  3,322 ServiceInstanceManager.java
12/07/99  04:26p                  9,858 ServiceEventHandler.java
12/07/99  04:26p                  2,093 ServiceInstanceManagerFailure.j
ava
12/07/99  04:26p                  2,142 ServiceControllerFailure.java
12/07/99  04:26p                  2,082 ServiceEventHandlerFailure.java
12/07/99  04:26p                  7,351 ServiceController.java
12/07/99  04:26p                  2,289 ServiceManagerIntf.esidl
12/07/99  04:26p                  5,091 ServiceManagerStub.java
12/07/99  04:26p                  1,273 ServiceManagerIntfMessageRegist

```

```

                                test
ry.java
12/07/99  04:26p                2,318 ServiceManagerIntf.java
12/07/99  04:26p                2,714 ServiceManagerImpl.java
12/06/00  04:33p                <DIR>          ui
                                17 File(s)      51,201 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\servicemanager\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:21p                809 Entries
12/16/99  06:21p                12 Entries.Log
12/16/99  06:21p                82 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                7 File(s)      1,795 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\servicemanager\ui

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/07/99  04:26p                1,888 ChangeStateFailed.java
12/06/00  04:33p                <DIR>          CVS
12/15/99  04:00p                842 Makefile
12/07/99  04:26p                2,036 ModelException.java
12/07/99  04:26p                6,867 ServiceManagerUI.java
12/07/99  04:26p                3,470 ServiceManagerTableModel.java
12/07/99  04:26p                15,099 ServiceManagerServlet.java
12/07/99  04:26p                21,904 ServiceManagerModel.java
12/07/99  04:26p                193 servlet.properties
12/07/99  04:26p                8,894 UiServiceManagerAdapter.java
12/07/99  04:26p                6,577 UiServiceManager.java
                                13 File(s)      67,770 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\management\servicemanager\ui\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:21p                589 Entries
12/16/99  06:21p                85 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                                6 File(s)      1,566 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se

test

rvices\management\web

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      servlets
          4 File(s)          0 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\web\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p              3 Entries
12/16/99  06:21p            18 Entries.Log
12/16/99  06:21p            71 Repository
12/16/99  06:21p            46 Root
12/16/99  06:21p           846 Template
          7 File(s)         984 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\web\servlets

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p      7,457 CoreServlet.java
12/06/00  04:33p      <DIR>      CVS
12/15/99  04:00p          461 Makefile
12/07/99  04:26p     10,758 ManagementServlet.java
          6 File(s)     18,676 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\management\web\servlets\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p            160 Entries
12/16/99  06:21p            80 Repository
12/16/99  06:21p            46 Root
12/16/99  06:21p           846 Template
          6 File(s)     1,132 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\se
rvices\tunnel

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p      6,943 AbstractSocket.java
```


12/07/99	04:26p		test
12/07/99	04:26p		1,894 AcceptedSocket.java
12/07/99	04:26p		2,219 ByteArray.java
12/07/99	04:26p		1,781 ClientSocketFactoryIF.java
12/07/99	04:26p		2,525 ClientSocketFactory.java
12/07/99	04:26p		2,428 ClientSocket.java
12/07/99	04:26p		700 compile.bat
12/07/99	04:26p		2,227 ConnectTunnelClientIF.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		6,227 Dialog.java
12/07/99	04:26p		3,218 ESClientSocketFactoryStub.java
12/07/99	04:26p		2,984 ESClientSocketFactorySkel.java
12/07/99	04:26p		1,777 ESClientSocketFactoryIF.java
12/07/99	04:26p		2,203 ESConnectTunnelClientIF.java
12/07/99	04:26p		4,777 ESConnectionFactory.java
12/07/99	04:26p		245 esdriver
12/07/99	04:26p		7,531 ESDriver.java
12/07/99	04:26p		3,970 ESSocketStub.java
12/07/99	04:26p		5,860 ESSocketSkel.java
12/07/99	04:26p		1,861 ESSocketIF.java
12/07/99	04:26p		3,210 ESTunnelServiceStub.java
12/07/99	04:26p		3,413 ESTunnelServiceSkel.java
12/07/99	04:26p		2,802 ESTunnelServiceIF.java
12/07/99	04:26p		1,114 Makefile
12/07/99	04:26p		0 Makefile.dep
12/07/99	04:26p		518 makefile.release
12/07/99	04:26p		1,724 ModelIF.java
12/07/99	04:26p		2,789 PassiveSocket.java
12/07/99	04:26p		2,195 SimpleConnectionFactory.java
12/07/99	04:26p		1,674 SocketRegistryIF.java
12/07/99	04:26p		1,686 SocketInternIF.java
12/07/99	04:26p		1,788 SocketIF.java
12/07/99	04:26p		252 tscmd
12/07/99	04:26p		4,534 TSCmd.java
12/07/99	04:26p		2,782 TunnelServiceIF.java
12/07/99	04:26p		6,878 TunnelService.java
12/07/99	04:26p		7,590 TunnelAttributes.java
		39 File(s)	106,319 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\services\tunnel\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		2,001 Entries
12/16/99	06:21p		63 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
		6 File(s)	2,956 bytes

test

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\ut
il

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		6,191 Assert.java
12/07/99	04:26p		2,023 Codecoverage.java
12/07/99	04:26p		2,289 CoverageThread.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		3,729 ErrorMsg.java
12/07/99	04:26p		15,301 ESAbstractWriter.java
12/07/99	04:26p		9,088 ESArray.java
12/07/99	04:26p		2,605 ESContainer.java
12/07/99	04:26p		26,220 ESDebug.java
12/07/99	04:26p		5,855 ESFileWriter.java
12/06/00	04:33p	<DIR>	esidl
12/07/99	04:26p		6,271 ESList.java
12/07/99	04:26p		3,817 ESLogStream.java
12/07/99	04:26p		2,997 ESMapNode.java
12/07/99	04:26p		16,204 ESMap.java
12/07/99	04:26p		2,747 ESPrintWriter.java
12/07/99	04:26p		4,619 ESSet.java
12/07/99	04:26p		6,346 ESString.java
12/07/99	04:26p		10,178 ESStrings.java
12/07/99	04:26p		17,634 ESText.java
12/07/99	04:26p		14,600 ESTRacer.java
12/07/99	04:26p		2,099 ExternalLogger.java
12/07/99	04:26p		6,221 FileServerServlet.java
12/07/99	04:26p		7,029 GenericListenerList.java
12/07/99	04:26p		19,890 IniFile.java
12/06/00	04:33p	<DIR>	io
12/15/99	04:00p		1,400 Makefile
12/06/00	04:33p	<DIR>	misc
12/07/99	04:26p		4,161 NullPrintStream.java
12/07/99	04:26p		4,350 Poll.java
12/07/99	04:26p		50,112 SysLoader.java
12/07/99	04:26p		6,639 TaskInJVM.java
12/07/99	04:26p		3,646 TaskLoader.java
12/07/99	04:26p		8,240 TaskNewJVM.java
12/07/99	04:26p		6,598 TaskRemote.java
12/07/99	04:26p		7,785 TaskReader.java
12/07/99	04:26p		4,939 TaskServer.java
12/07/99	04:26p		3,170 TaskWatcher.java
12/07/99	04:26p		32,665 Task.java
12/07/99	04:26p		3,430 TestHelper.java
12/07/99	04:26p		6,416 Trace.java
12/06/00	04:33p	<DIR>	ts

```

                                test
12/06/00  04:33p      <DIR>      tsutil
12/07/99  04:26p      6,296 UniqueID.java
                        46 File(s) 343,800 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p      1,957 Entries
12/16/99  06:21p      69 Entries.Log
12/16/99  06:21p      52 Repository
12/16/99  06:21p      46 Root
12/16/99  06:21p      846 Template
                        7 File(s) 2,970 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\esidl

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p      11,106 CodeGen.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p      32,286 IDLCompiler.java
12/07/99  04:26p      13,857 IDLTypeRepository.java
12/07/99  04:26p      3,997 ImportedTypeRepository.java
12/07/99  04:26p      2,391 InvalidIDLTypeException.java
12/07/99  04:26p      593 Makefile
12/07/99  04:26p      20,659 Marshaller.java
12/07/99  04:26p      3,320 MyPrintWriter.java
12/07/99  04:26p      7,153 PrimitiveTypeMap.java
12/07/99  04:26p      18,662 StubCompiler.java
12/07/99  04:26p      18,593 TypeValidator.java
12/07/99  04:26p      14,618 Utils.java
                        15 File(s) 147,235 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\esidl\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p      659 Entries
12/16/99  06:21p      58 Repository
12/16/99  06:21p      46 Root
12/16/99  06:21p      846 Template
                        6 File(s) 1,609 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\ut

test

il\io

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                5,283 FileOutputStream.java
12/07/99  04:26p                4,154 LogContext.java
12/07/99  04:26p                8,302 LogEnv.java
12/07/99  04:26p                7,550 Logger.java
12/07/99  04:26p                4,571 LogOutputStream.java
12/07/99  04:26p                455 Makefile
12/07/99  04:26p                3,029 NullOutputStream.java
                10 File(s)        33,344 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\ut
il\io\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                369 Entries
12/16/99  06:21p                55 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
                6 File(s)        1,316 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\ut
il\misc

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p                2,422 Array.java
12/07/99  04:26p            12,460 ByteArray.java
12/07/99  04:26p                4,536 ByteBuffer.java
12/07/99  04:26p            10,566 Cons.java
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                429 Makefile
12/07/99  04:26p            5,387 Queue.java
12/07/99  04:26p            2,042 Testable.java
                10 File(s)        37,842 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\ut
il\misc\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                341 Entries
12/16/99  06:21p                57 Repository
12/16/99  06:21p                46 Root
```

```

                                test
12/16/99  06:21p                846 Template
                                1,290 bytes
        6 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\ts

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                731 Makefile
12/07/99  04:26p                3,820 PrettyPrintManager.java
12/07/99  04:26p               49,630 TSAtomic.java
12/07/99  04:26p                5,305 TSBoolean.java
12/07/99  04:26p                2,867 TSDatePP.java
12/07/99  04:26p                2,764 TSDefaultFactory.java
12/07/99  04:26p                3,105 TSFactory.java
12/07/99  04:26p               10,751 TSLexical.java
12/07/99  04:26p               18,143 TSList.java
12/07/99  04:26p                2,829 TSMapEnumeration.java
12/07/99  04:26p               28,181 TSMAP.java
12/07/99  04:26p                3,044 TSObservable.java
12/07/99  04:26p               22,612 TSParser.java
12/07/99  04:26p                2,237 TSParseException.java
12/07/99  04:26p                2,488 TSPrettyPrinter.java
12/07/99  04:26p                6,701 TSTest.java
12/07/99  04:26p               30,139 TSUtils.java
12/07/99  04:26p                8,179 TSValue.java
12/07/99  04:26p               3,771 TSVisitor.java
12/07/99  04:26p               2,520 TypeErrorException.java
        23 File(s)               209,817 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\ts\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:21p                1,048 Entries
12/16/99  06:21p                55 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p                846 Template
        6 File(s)                1,995 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\tsutil

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS

```

12/07/99	04:26p	test
12/07/99	04:26p	4,605 List.java
12/07/99	04:26p	498 Makefile
12/07/99	04:26p	4,889 Map.java
12/07/99	04:26p	11,159 Syn.java
12/07/99	04:26p	2,545 TSBinary.java
12/07/99	04:26p	2,335 TSBoolean.java
12/07/99	04:26p	2,342 TSInteger.java
12/07/99	04:26p	2,310 TSNumber.java
12/07/99	04:26p	2,225 TSString.java
12/07/99	04:26p	2,165 TSTag.java
12/07/99	04:26p	2,426 TSToken.java
	14 File(s)	37,499 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\util\tsutil\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		532 Entries
12/16/99	06:21p		59 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
	6 File(s)		1,483 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\webaccess

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	conf
12/07/99	04:26p		3,264 ConverterIntf.java
12/07/99	04:26p		2,913 ConverterInfo.java
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		3,765 DispatchTable.java
12/07/99	04:26p		15,409 ESConverter.java
12/07/99	04:26p		10,988 ESFormMapper.java
12/07/99	04:26p		11,250 ESMapper.java
12/07/99	04:26p		2,067 ESMessageXml.java
12/07/99	04:26p		5,440 ESProcessor.java
12/07/99	04:26p		4,423 FormGenericMapper.java
12/07/99	04:26p		4,589 FormLoginMapper.java
12/07/99	04:26p		3,703 GenericMapper.java
12/07/99	04:26p		6,801 GenericConverter.java
12/06/00	04:33p	<DIR>	htdocs
12/07/99	04:26p		4,429 Mailbox.java
12/15/99	04:00p		1,385 Makefile
12/07/99	04:26p		634 makefile.release
12/07/99	04:26p		3,165 MapperInfo.java

```

12/07/99 04:26p      test
12/07/99 04:26p      2,573 MapperIntf.java
12/07/99 04:26p      2,040 ProcessorIntf.java
12/07/99 04:26p      2,716 ProcessorInfo.java
12/07/99 04:26p      3,667 Queue.java
12/07/99 04:26p      2,257 RequestBadRequest.java
12/07/99 04:26p      2,288 RequestNotFound.java
12/07/99 04:26p      2,265 RequestNoContent.java
12/07/99 04:26p      2,267 RequestMultipleChoice.java
12/07/99 04:26p      2,241 RequestForbidden.java
12/07/99 04:26p      4,866 ServiceDaemon.java
12/07/99 04:26p      4,779 ServiceTranslator.java
12/07/99 04:26p      3,155 ServiceTable.java
12/07/99 04:26p      7,992 ServiceHandler.java
12/07/99 04:26p      16,885 ServiceDispatcher.java
12/07/99 04:26p      5,929 SessionTable.java
12/07/99 04:26p      2,971 Session.java
12/07/99 04:26p      3,732 TranslationTable.java
12/07/99 04:26p      5,255 URLBinder.java
12/07/99 04:26p      3,455 URLCache.java
12/07/99 04:26p      7,845 URLConverter.java
12/07/99 04:26p      6,889 Util.java
12/07/99 04:26p      2,663 WAExceptionInfo.java
12/07/99 04:26p      2,472 WANameList.java
12/07/99 04:26p      6,942 WebAccess.java
12/07/99 04:26p      9,263 XMLApp3.java
                                46 File(s)      201,632 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\we
baccess\conf

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      27,318 httpd.conf
12/06/00 04:33p      <DIR>      jserv
                                5 File(s)      27,318 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\we
baccess\conf\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:21p      50 Entries
12/16/99 06:21p      15 Entries.Log
12/16/99 06:21p      62 Repository
12/16/99 06:21p      46 Root
12/16/99 06:21p      846 Template
                                7 File(s)      1,019 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\we
baccess\conf\jserv

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		6,559 jserv.conf
12/07/99	04:26p		12,790 jserv.properties
12/07/99	04:26p		6,496 zone.properties
6 File(s)			25,845 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\we
baccess\conf\jserv\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		155 Entries
12/16/99	06:21p		68 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
6 File(s)			1,115 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\we
baccess\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:21p		2,218 Entries
12/16/99	06:21p		30 Entries.Log
12/16/99	06:21p		57 Repository
12/16/99	06:21p		46 Root
12/16/99	06:21p		846 Template
7 File(s)			3,197 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\we
baccess\htdocs

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		699 ask-lu-vocab.html
12/07/99	04:26p		703 ask-reg-vocab.html
12/07/99	04:26p		665 book-buying.html
12/07/99	04:26p		509 button-back-to-register.gif
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		11,638 ESPEAK.jpg
12/07/99	04:26p		886 green-ball.gif
12/07/99	04:26p		1,454 index.html


```

                                test
12/07/99  04:26p                1,107 login-reply.html
12/07/99  04:26p                1,102 logo.gif
12/07/99  04:26p                1,265 lookup-form.html
12/07/99  04:26p            691,318 may10ic.bmp
12/07/99  04:26p            41,279 may10ic.jpg
12/07/99  04:26p                1,522 register-form.html
12/06/00  04:33p        <DIR>          servlets
12/07/99  04:26p            148,992 web-espeak_slides.ppt
12/07/99  04:26p                5,346 welcomenew.gif
                                908,485 bytes
                                19 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\webaccess\htdocs\CVS

```

12/06/00  04:33p        <DIR>          .
12/06/00  04:33p        <DIR>          ..
12/16/99  06:22p                785 Entries
12/16/99  06:22p                18 Entries.Log
12/16/99  06:21p                64 Repository
12/16/99  06:21p                46 Root
12/16/99  06:21p            846 Template
                                1,759 bytes
                                7 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\webaccess\htdocs\servlets

```

12/06/00  04:33p        <DIR>          .
12/06/00  04:33p        <DIR>          ..
12/07/99  04:26p            20,614 BookBroker.java
12/06/00  04:33p        <DIR>          CVS
12/07/99  04:26p            7,045 Forms.java
12/07/99  04:26p            6,645 HTML2XML.java
12/07/99  04:26p            4,089 IsItWorking.java
12/15/99  04:00p            575 Makefile
12/07/99  04:26p            40 proxy.cfg
12/07/99  04:26p            10,518 ProxyBarnes.java
12/07/99  04:26p            12,489 ProxyFatBrain.java
12/07/99  04:26p            3,578 ProxyInfo.java
12/07/99  04:26p            5,467 ShowRegistrationForm.java
12/07/99  04:26p            2,224 XMLDocTags.java
                                73,284 bytes
                                14 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\java\net\espeak\webaccess\htdocs\servlets\CVS

```

12/06/00  04:33p        <DIR>          .
12/06/00  04:33p        <DIR>          ..
12/16/99  06:22p                569 Entries

```

```

                                test
12/16/99  06:22p                73 Repository
12/16/99  06:22p                46 Root
12/16/99  06:22p                846 Template
                                1,534 bytes
        6 File(s)

```

Directory of E:\e-speak-src_991217\platform\ES\src\perl

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/06/00  04:33p      <DIR>      plesi
        4 File(s)                0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\perl\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:22p                3 Entries
12/16/99  06:22p            15 Entries.Log
12/16/99  06:22p            36 Repository
12/16/99  06:22p            46 Root
12/16/99  06:22p            846 Template
        7 File(s)            946 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\perl\plesi

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p            9,832 ABI.pm
12/07/99  04:26p           10,309 abi2c.pl
12/07/99  04:26p            2,220 abi2txt.pl
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p           18,584 ESLib.pm
12/07/99  04:26p           13,760 ESReader.pm
12/07/99  04:26p            8,467 ESUtil.pm
12/07/99  04:26p            7,139 ESWriter.pm
12/07/99  04:26p            7,051 logexpand.pl
12/07/99  04:26p           15,799 README.txt
12/07/99  04:26p            2,636 rtc.pl
12/07/99  04:26p            4,849 rts.pl
12/07/99  04:26p            2,019 sigbytes.pl
12/07/99  04:26p            2,701 tc.pl
12/07/99  04:26p            3,179 ts.pl
        17 File(s)          108,545 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\perl\plesi\CVS

```

12/06/00  04:33p      <DIR>      .

```

```

                                test
12/06/00  04:33p          <DIR>      ..
12/16/99  06:22p                      639 Entries
12/16/99  06:22p                      42 Repository
12/16/99  06:22p                      46 Root
12/16/99  06:22p                      846 Template
                                6 File(s)      1,573 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\python

```

12/06/00  04:33p          <DIR>      .
12/06/00  04:33p          <DIR>      ..
12/06/00  04:33p          <DIR>      CVS
12/06/00  04:33p          <DIR>      pyesi
                                4 File(s)      0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\python\CVS

```

12/06/00  04:33p          <DIR>      .
12/06/00  04:33p          <DIR>      ..
12/16/99  06:22p                      3 Entries
12/16/99  06:22p                      15 Entries.Log
12/16/99  06:22p                      38 Repository
12/16/99  06:22p                      46 Root
12/16/99  06:22p                      846 Template
                                7 File(s)      948 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi

```

12/06/00  04:33p          <DIR>      .
12/06/00  04:33p          <DIR>      ..
12/06/00  04:33p          <DIR>      abi
12/06/00  04:33p          <DIR>      CVS
12/06/00  04:33p          <DIR>      io
12/06/00  04:33p          <DIR>      net
12/07/99  04:26p                      25,590 README.txt
12/06/00  04:33p          <DIR>      util
12/07/99  04:26p                      0 __init__.py
                                9 File(s)      25,590 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\abi

```

12/06/00  04:33p          <DIR>      .
12/06/00  04:33p          <DIR>      ..
12/07/99  04:26p                      17,585 abi.py
12/07/99  04:26p                      2,480 abi2txt.py
12/07/99  04:26p                      48,218 abiobj.py
12/06/00  04:33p          <DIR>      CVS
12/07/99  04:26p                      18,088 eslib.py

```

12/07/99	04:26p		test
12/07/99	04:26p		6,762 esobj.py
12/07/99	04:26p		2,476 esutil.py
12/07/99	04:26p		9,925 es_input_stream.py
12/07/99	04:26p		11,082 es_output_stream.py
12/07/99	04:26p		10,554 genabi.py
12/07/99	04:26p		7,026 genobj.py
12/06/00	04:33p	<DIR>	msg
12/07/99	04:26p		5,049 rtc.py
12/07/99	04:26p		10,667 rts.py
12/07/99	04:26p		2,168 sigbytes.py
12/07/99	04:26p		5,455 tc.py
12/07/99	04:26p		7,777 ts.py
12/07/99	04:26p		0 __init__.py
20 File(s)			165,312 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\abi\C
VS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:22p		744 Entries
12/16/99	06:22p		13 Entries.Log
12/16/99	06:22p		48 Repository
12/16/99	06:22p		46 Root
12/16/99	06:22p		846 Template
7 File(s)			1,697 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\abi\msg

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		8 msg0001.ser
12/07/99	04:26p		117 msg0002.ser
12/07/99	04:26p		154 msg0003.ser
12/07/99	04:26p		53 msg0004.ser
12/07/99	04:26p		159 msg0005.ser
12/07/99	04:26p		53 msg0006.ser
12/07/99	04:26p		194 msg0007.ser
12/07/99	04:26p		53 msg0008.ser
12/07/99	04:26p		151 msg0009.ser
12/07/99	04:26p		53 msg0010.ser
12/07/99	04:26p		353 msg0011.ser
12/07/99	04:26p		53 msg0012.ser
12/07/99	04:26p		275 msg0013.ser
12/07/99	04:26p		53 msg0014.ser
12/07/99	04:26p		8 msg0015.ser

12/07/99	04:26p	test	
12/07/99	04:26p	117	msg0016.ser
12/07/99	04:26p	154	msg0017.ser
12/07/99	04:26p	53	msg0018.ser
12/07/99	04:26p	159	msg0019.ser
12/07/99	04:26p	53	msg0020.ser
12/07/99	04:26p	136	msg0021.ser
12/07/99	04:26p	54	msg0022.ser
12/07/99	04:26p	92	msg0023.ser
12/07/99	04:26p	54	msg0024.ser
12/07/99	04:26p	157	msg0025.ser
12/07/99	04:26p	53	msg0026.ser
12/07/99	04:26p	91	msg0027.ser
12/07/99	04:26p	57	msg0028.ser
12/07/99	04:26p	71	msg0029.ser
12/07/99	04:26p	54	msg0030.ser
12/07/99	04:26p	71	msg0031.ser
12/07/99	04:26p	54	msg0032.ser
12/07/99	04:26p	86	msg0033.ser
12/07/99	04:26p	54	msg0034.ser
12/07/99	04:26p	156	msg0035.ser
12/07/99	04:26p	54	msg0036.ser
12/07/99	04:26p	92	msg0037.ser
12/07/99	04:26p	54	msg0038.ser
12/07/99	04:26p	157	msg0039.ser
12/07/99	04:26p	53	msg0040.ser
12/07/99	04:26p	91	msg0041.ser
12/07/99	04:26p	57	msg0042.ser
12/07/99	04:26p	71	msg0043.ser
12/07/99	04:26p	54	msg0044.ser
12/07/99	04:26p	71	msg0045.ser
12/07/99	04:26p	54	msg0046.ser
12/07/99	04:26p	86	msg0047.ser
12/07/99	04:26p	54	msg0048.ser
12/07/99	04:26p	121	msg0049.ser
12/07/99	04:26p	145	msg0050.ser
12/07/99	04:26p	75	msg0051.ser
12/07/99	04:26p	93	msg0052.ser
55 File(s)		4,895 bytes	

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\abi\msg\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:22p		2,499 Entries
12/16/99	06:22p		52 Repository
12/16/99	06:22p		46 Root
12/16/99	06:22p		846 Template

test
6 File(s) 3,443 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:22p		98 Entries
12/16/99	06:22p		52 Entries.Log
12/16/99	06:22p		44 Repository
12/16/99	06:22p		46 Root
12/16/99	06:22p		846 Template
7 File(s)			1,086 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\io

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		6,758 array_buffer.py
12/07/99	04:26p		3,326 buffered_output_stream.py
12/07/99	04:26p		2,691 console_output_stream.py
12/07/99	04:26p		2,762 console_input_stream.py
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		7,225 data_input_stream.py
12/07/99	04:26p		9,129 data_output_stream.py
12/07/99	04:26p		5,792 demultiplexer.py
12/07/99	04:26p		2,668 error_output_stream.py
12/07/99	04:26p		4,906 file.py
12/07/99	04:26p		3,843 file_append_stream.py
12/07/99	04:26p		3,453 file_input_stream.py
12/07/99	04:26p		3,695 file_output_stream.py
12/07/99	04:26p		2,704 filter_output_stream.py
12/07/99	04:26p		2,678 filter_input_stream.py
12/07/99	04:26p		6,055 input_stream.py
12/07/99	04:26p		4,126 message_copier.py
12/07/99	04:26p		4,358 message_output_stream.py
12/07/99	04:26p		3,948 message_input_stream.py
12/07/99	04:26p		4,315 multiplex_output_stream.py
12/07/99	04:26p		4,087 multiplex_input_stream.py
12/07/99	04:26p		6,477 output_stream.py
12/07/99	04:26p		3,535 queue_output_stream.py
12/07/99	04:26p		3,579 queue_input_stream.py
12/07/99	04:26p		4,494 router.py
12/07/99	04:26p		4,713 routing_output_stream.py
12/07/99	04:26p		4,239 routing_input_stream.py
12/07/99	04:26p		3,466 string_buffer_output_stream.py
12/07/99	04:26p		3,646 string_buffer_input_stream.py
12/07/99	04:26p		8,115 string_buffer.py
12/07/99	04:26p		0 __init__.py

test
33 File(s) 130,783 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\io\CV
S

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:22p		1,724 Entries
12/16/99	06:22p		47 Repository
12/16/99	06:22p		46 Root
12/16/99	06:22p		846 Template
6 File(s)			2,663 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\net

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		14,763 client_socket.py
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		9,316 http_client_stream.py
12/07/99	04:26p		5,148 server_socket.py
12/07/99	04:26p		4,848 simple_server.py
12/07/99	04:26p		3,626 simple_handler.py
12/07/99	04:26p		3,266 socket_output_stream.py
12/07/99	04:26p		3,294 socket_input_stream.py
12/07/99	04:26p		4,710 threaded_server.py
12/07/99	04:26p		2,562 threaded_handler.py
12/07/99	04:26p		0 __init__.py
13 File(s)			51,533 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\net\C
VS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:22p		552 Entries
12/16/99	06:22p		48 Repository
12/16/99	06:22p		46 Root
12/16/99	06:22p		846 Template
6 File(s)			1,492 bytes

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\util

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		10,033 file_observer.py
12/07/99	04:26p		16,433 file_queue.py

```

12/07/99 04:26p      test
12/07/99 04:26p      8,638 log.py
12/07/99 04:26p      3,565 thread_pool.py
12/07/99 04:26p      3,039 timestamp.py
12/07/99 04:26p      6,417 url.py
12/07/99 04:26p      0 __init__.py
      10 File(s)      48,125 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\src\python\pyesi\util\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:22p      340 Entries
12/16/99 06:22p      49 Repository
12/16/99 06:22p      46 Root
12/16/99 06:22p      846 Template
      6 File(s)      1,281 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      225 Makefile
12/06/00 04:33p      <DIR>      sharebroker
12/06/00 04:33p      <DIR>      usedcarsale
      6 File(s)      225 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:22p      48 Entries
12/16/99 06:23p      42 Entries.Log
12/16/99 06:22p      36 Repository
12/16/99 06:22p      46 Root
12/16/99 06:22p      846 Template
      7 File(s)      1,018 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p      459 compile.bat
12/06/00 04:33p      <DIR>      config
12/06/00 04:33p      <DIR>      CVS
12/06/00 04:33p      <DIR>      doc
12/07/99 04:26p      296 Makefile

```



```

12/07/99  04:26p          test
12/07/99  04:26p          2,040 setenv
12/07/99  04:26p          1,416 setenv.bat
12/06/00  04:33p          <DIR>          src
                        10 File(s)          4,211 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/06/00  04:33p          <DIR>          CVS
12/06/00  04:33p          <DIR>          example1
12/06/00  04:33p          <DIR>          example2
12/06/00  04:33p          <DIR>          example3
12/06/00  04:33p          <DIR>          example4
12/06/00  04:33p          <DIR>          example5
                        8 File(s)          0 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\CVS

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/16/99  06:22p          3 Entries
12/16/99  06:22p          90 Entries.Log
12/16/99  06:22p          55 Repository
12/16/99  06:22p          46 Root
12/16/99  06:22p          846 Template
                        7 File(s)          1,040 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example1

```

12/06/00  04:33p          <DIR>          .
12/06/00  04:33p          <DIR>          ..
12/07/99  04:26p          20 bank.pr
12/07/99  04:26p          261 bankvocab.ini
12/07/99  04:26p          1,247 BankVocabulary.xml
12/06/00  04:33p          <DIR>          CVS
12/07/99  04:26p          70 runbankvocab.bat
12/07/99  04:26p          61 runbankvocab.sh
12/07/99  04:26p          110 runvocabfinder.bat
12/07/99  04:26p          100 runvocabfinder.sh
                        10 File(s)          1,869 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example1\CVS

```

                                test
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:22p                      366 Entries
12/16/99  06:22p                      64 Repository
12/16/99  06:22p                      46 Root
12/16/99  06:22p                      846 Template
                                6 File(s)      1,322 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example2

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p                      290 bank.ini
12/07/99  04:26p                      18 bank.pr
12/07/99  04:26p                      1,322 BankService.xml
12/07/99  04:26p                      1,247 BankVocabulary.xml
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                      65 runbankservice.bat
12/07/99  04:26p                      90 runbankservice.sh
12/07/99  04:26p                      107 runclient.bat
12/07/99  04:26p                      98 runclient.sh
                                11 File(s)      3,237 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example2\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:22p                      407 Entries
12/16/99  06:22p                      64 Repository
12/16/99  06:22p                      46 Root
12/16/99  06:22p                      846 Template
                                6 File(s)      1,363 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example3

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p                      6 account.dat
12/07/99  04:26p                      314 addfunds.gif
12/07/99  04:26p                      1,097 add_share.gif
12/07/99  04:26p                      18 bank.pr
12/07/99  04:26p                      1,322 BankService.xml
12/07/99  04:26p                      1,247 BankVocabulary.xml
12/07/99  04:26p                      1,091 buy.gif
12/07/99  04:26p                      131 cancelord.gif

```

```

12/07/99 04:26p      test
12/07/99 04:26p      260 change_dir.gif
12/07/99 04:26p      254 connect.gif
12/07/99 04:26p      330 cpwd.gif
12/07/99 04:26p      320 create_vb.gif
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      131 delete.gif
12/07/99 04:26p      182 disconnect.gif
12/07/99 04:26p      275 download.gif
12/07/99 04:26p      110 folder.gif
12/07/99 04:26p      174 help.gif
12/07/99 04:26p      423 localserver.ini
12/07/99 04:26p      300 login.gif
12/07/99 04:26p      1,189 open_act.gif
12/07/99 04:26p      213 reconnect.gif
12/07/99 04:26p      72 runserver.bat
12/07/99 04:26p      130 runserver.sh
12/07/99 04:26p      1,094 sell.gif
12/07/99 04:26p      918 ShareBrokerVocabulary.xml
12/07/99 04:26p      1,178 ShareBrokerService.xml
12/07/99 04:26p      95 starttrader.sh
12/07/99 04:26p      71 stocktrade.bat
12/07/99 04:26p      1,170 title.gif
12/07/99 04:26p      20 trader.pro
12/07/99 04:26p      1,335 Trader.xml
12/07/99 04:26p      20 traderIntf.pro
12/07/99 04:26p      1,253 TraderVocabulary.xml
12/07/99 04:26p      1,118 TraderService.xml
12/07/99 04:26p      314 upload.gif
12/07/99 04:26p      285 watch.gif
      39 File(s)      18,460 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\c
onfig\example3\CVS

```

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:22p      1,787 Entries
12/16/99 06:22p      64 Repository
12/16/99 06:22p      46 Root
12/16/99 06:22p      846 Template
      6 File(s)      2,743 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\c
onfig\example4

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:26p      6 account.dat

```

```

12/07/99 04:26p      test
12/07/99 04:26p      314 addfunds.gif
12/07/99 04:26p      1,097 add_share.gif
12/07/99 04:26p      1,444 BankService.xml
12/07/99 04:26p      1,362 BankVocabulary.xml
12/07/99 04:26p      1,091 buy.gif
12/07/99 04:26p      131 cancelord.gif
12/07/99 04:26p      260 change_dir.gif
12/07/99 04:26p      254 connect.gif
12/07/99 04:26p      330 cpwd.gif
12/07/99 04:26p      320 create_vb.gif
12/06/00 04:33p      <DIR>      CVS
12/07/99 04:26p      131 delete.gif
12/07/99 04:26p      182 disconnect.gif
12/07/99 04:26p      275 download.gif
12/07/99 04:26p      110 folder.gif
12/07/99 04:26p      174 help.gif
12/06/00 04:33p      <DIR>      host1
12/06/00 04:33p      <DIR>      host2
12/07/99 04:26p      300 login.gif
12/07/99 04:26p      1,189 open_act.gif
12/07/99 04:26p      213 reconnect.gif
12/07/99 04:26p      1,094 sell.gif
12/07/99 04:26p      1,485 ShareBrokerVocabulary.xml
12/07/99 04:26p      1,429 ShareBrokerService2.xml
12/07/99 04:26p      1,430 ShareBrokerService1.xml
12/06/00 04:33p      <DIR>      singlehost
12/07/99 04:26p      1,170 title.gif
12/07/99 04:26p      1,335 Trader.xml
12/07/99 04:26p      1,253 TraderVocabulary.xml
12/07/99 04:26p      1,118 TraderService.xml
12/07/99 04:26p      314 upload.gif
12/07/99 04:26p      285 watch.gif
                                35 File(s)
                                20,096 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:22p      1,453 Entries
12/16/99 06:22p      50 Entries.Log
12/16/99 06:22p      64 Repository
12/16/99 06:22p      46 Root
12/16/99 06:22p      846 Template
                                7 File(s)
                                2,459 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\host1

test

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p                141 bank.pr
12/07/99  04:26p                22 col
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p            114 runbroker1.bat
12/07/99  04:26p            138 runbroker1.sh
12/07/99  04:26p             73 sharebroker1.bat
12/07/99  04:26p            672 sharebroker1.ini
12/07/99  04:26p            304 sharebroker1.pr
12/07/99  04:26p            158 sharebroker1.sh
12/07/99  04:26p            139 stocktrade.pr
      12 File(s)                1,761 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\host1\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:22p            448 Entries
12/16/99  06:22p             70 Repository
12/16/99  06:22p             46 Root
12/16/99  06:22p            846 Template
      6 File(s)                1,410 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\host2

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:26p            141 bank.pr
12/07/99  04:26p             22 co2
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p            115 runbroker2.bat
12/07/99  04:26p            137 runbroker2.sh
12/07/99  04:26p             73 sharebroker2.bat
12/07/99  04:26p            730 sharebroker2.ini
12/07/99  04:26p            302 sharebroker2.pr
12/07/99  04:26p            177 sharebroker2.sh
12/07/99  04:26p            139 stocktrade.pr
      12 File(s)                1,836 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\host2\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..

```

```

                                test
12/16/99  06:22p                448 Entries
12/16/99  06:22p                70 Repository
12/16/99  06:22p                46 Root
12/16/99  06:22p                846 Template
                                6 File(s)          1,410 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\singlehost

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/07/99  04:26p                43 bank.pr
12/06/00  04:33p                <DIR>          CVS
12/07/99  04:26p                549 localserver.ini
12/07/99  04:26p                115 runbroker2.bat
12/07/99  04:26p                115 runbroker1.bat
12/07/99  04:26p                72 runserver.bat
12/07/99  04:26p                206 sharebroker2.pr
12/07/99  04:26p                207 sharebroker1.pr
12/07/99  04:26p                71 stocktrade.bat
12/07/99  04:26p                43 stocktrade.pr
                                12 File(s)        1,421 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example4\singlehost\CVS

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/16/99  06:22p                456 Entries
12/16/99  06:22p                75 Repository
12/16/99  06:22p                46 Root
12/16/99  06:22p                846 Template
                                6 File(s)          1,423 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\config\example5

```

12/06/00  04:33p                <DIR>          .
12/06/00  04:33p                <DIR>          ..
12/07/99  04:26p                6 account.dat
12/07/99  04:26p                314 addfunds.gif
12/07/99  04:26p                1,097 add_share.gif
12/07/99  04:26p                43 bank.pr
12/07/99  04:26p                1,444 BankService.xml
12/07/99  04:26p                1,362 BankVocabulary.xml
12/07/99  04:26p                1,091 buy.gif
12/07/99  04:26p                131 cancelord.gif
12/07/99  04:26p                260 change_dir.gif

```



```

                                test
12/07/99  04:26p                1,253 TraderVocabulary.xml
12/07/99  04:26p                1,118 TraderService.xml
12/07/99  04:26p                 314 upload.gif
12/07/99  04:26p                 285 watch.gif
                                63 File(s)      220,512 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\c
onfig\example5\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:22p                2,999 Entries
12/16/99  06:22p                 64 Repository
12/16/99  06:22p                 46 Root
12/16/99  06:22p                846 Template
                                6 File(s)      3,955 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\C
VS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:22p                186 Entries
12/16/99  06:23p                 42 Entries.Log
12/16/99  06:22p                 48 Repository
12/16/99  06:22p                 46 Root
12/16/99  06:22p                846 Template
                                7 File(s)      1,168 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\d
oc

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:26p                1,806,132 ShareBrokerDocs.zip
                                4 File(s)      1,806,132 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\d
oc\CVS

```

12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:23p                59 Entries
12/16/99  06:22p                 52 Repository
12/16/99  06:22p                 46 Root
12/16/99  06:22p                846 Template
                                6 File(s)      1,003 bytes

```


test

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/06/00	04:33p	<DIR>	CVS
12/06/00	04:33p	<DIR>	example1
12/06/00	04:33p	<DIR>	example2
12/06/00	04:33p	<DIR>	example3
12/06/00	04:33p	<DIR>	example4
12/06/00	04:33p	<DIR>	example5
8 File(s)			0 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:23p		3 Entries
12/16/99	06:23p		90 Entries.Log
12/16/99	06:23p		52 Repository
12/16/99	06:23p		46 Root
12/16/99	06:23p		846 Template
7 File(s)			1,037 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example1

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:26p		4,723 BankVocabulary.java
12/07/99	04:26p		4,109 BankVocabFinder.java
12/07/99	04:26p		353 compile.bat
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:26p		361 Makefile
7 File(s)			9,546 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example1\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:23p		209 Entries
12/16/99	06:23p		61 Repository
12/16/99	06:23p		46 Root
12/16/99	06:23p		846 Template
6 File(s)			1,162 bytes

test

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example2

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:27p                3,326 AccountNumberGenerator.java
12/07/99  04:27p                1,930 Address.esidl
12/07/99  04:27p                6,326 BankClient.java
12/07/99  04:27p                2,748 BankServiceIntf.esidl
12/07/99  04:27p                3,192 BankServiceIntfFinder.java
12/07/99  04:27p            15,603 BankServiceImpl.java
12/07/99  04:27p                4,094 BankService.java
12/07/99  04:27p                4,731 BankVocabulary.java
12/07/99  04:27p                1,823 Company.esidl
12/07/99  04:27p                491 compile.bat
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:27p                1,007 Makefile
12/07/99  04:27p                2,063 OrderDetailParam.esidl
12/07/99  04:27p                2,275 Semaphore.java
12/07/99  04:27p                2,003 ShareDetailParam.esidl
12/07/99  04:27p                3,223 ShareDetail.java
12/07/99  04:27p                2,461 UserAccount.java
                                19 File(s)
                                57,296 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example2\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:23p                874 Entries
12/16/99  06:23p                61 Repository
12/16/99  06:23p                46 Root
12/16/99  06:23p                846 Template
                                6 File(s)
                                1,827 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example3

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:27p                3,325 AccountNumberGenerator.java
12/07/99  04:27p                5,773 AddFundsGUI.java
12/07/99  04:27p                1,930 Address.esidl
12/07/99  04:27p                8,140 AddSharesGUI.java
12/07/99  04:27p            10,909 AuthorizationWindow.java
12/07/99  04:27p                2,783 BankServiceIntf.esidl
12/07/99  04:27p                3,119 BankServiceIntfFinder.java
```

12/07/99	04:27p		test
12/07/99	04:27p		14,971 BankServiceImpl.java
12/07/99	04:27p		4,083 BankService.java
12/07/99	04:27p		4,723 BankVocabulary.java
12/07/99	04:27p		18,119 BrokerLogic.java
12/07/99	04:27p		10,571 BuySharesGUI.java
12/07/99	04:27p		7,170 ChangePasswordGUI.java
12/07/99	04:27p		1,819 Company.esidl
12/07/99	04:27p		689 compile.bat
12/06/00	04:33p	<DIR>	CVS
12/07/99	04:27p		9,308 ListAccountGUI.java
12/07/99	04:27p		18,944 ListSharesGUI.java
12/07/99	04:27p		7,871 LoginGUI.java
12/15/99	04:00p		2,405 Makefile
12/07/99	04:27p		1,901 MatchThread.java
12/07/99	04:27p		5,656 MessageDialog.java
12/07/99	04:27p		7,592 OpenAccountGUI.java
12/07/99	04:27p		2,063 OrderDetailParam.esidl
12/07/99	04:27p		3,610 OrderDetail.java
12/07/99	04:27p		9,686 SellSharesGUI.java
12/07/99	04:27p		2,279 Semaphore.java
12/07/99	04:27p		2,650 ServiceDeployer.java
12/07/99	04:27p		2,225 ShareBrokerIntf.esidl
12/07/99	04:27p		3,256 ShareBrokerIntfFinder.java
12/07/99	04:27p		4,600 ShareBrokerImpl.java
12/07/99	04:27p		2,046 ShareDetailParam.esidl
12/07/99	04:27p		3,223 ShareDetail.java
12/07/99	04:27p		4,449 StartShareBrokerService.java
12/07/99	04:27p		6,322 StartTraderService.java
12/07/99	04:27p		23,802 StockTrade.java
12/07/99	04:27p		1,921 TraderQuerySpec.esidl
12/07/99	04:27p		4,840 TraderVocabulary.java
12/07/99	04:27p		1,948 TraderIntf.esidl
12/07/99	04:27p		2,731 TraderService.java
12/07/99	04:27p		4,384 TraderIntfFinder.java
12/07/99	04:27p		2,979 TraderImpl.java
12/07/99	04:27p		1,969 Transaction.esidl
12/07/99	04:27p		2,459 UserAccount.java
		46 File(s)	245,243 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example3\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:23p		2,389 Entries
12/16/99	06:23p		61 Repository
12/16/99	06:23p		46 Root
12/16/99	06:23p		846 Template

6 File(s)

test
3,342 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example4

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:27p           3,407 AccountNumberGenerator.java
12/07/99  04:27p           5,829 AddFundsGUI.java
12/07/99  04:27p           1,930 Address.esidl
12/07/99  04:27p           8,196 AddSharesGUI.java
12/07/99  04:27p        10,880 AuthorizationWindow.java
12/07/99  04:27p           2,783 BankServiceIntf.esidl
12/07/99  04:27p           6,052 BankServiceIntfFinder.java
12/07/99  04:27p        17,695 BankServiceImpl.java
12/07/99  04:27p           4,392 BankService.java
12/07/99  04:27p           4,718 BankVocabulary.java
12/07/99  04:27p        33,103 BrokerLogic.java
12/07/99  04:27p        10,530 BuySharesGUI.java
12/07/99  04:27p           5,975 CancelOrderGUI.java
12/07/99  04:27p           7,389 ChangePasswordGUI.java
12/07/99  04:27p           1,819 Company.esidl
12/07/99  04:27p           691 compile.bat
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:27p           2,049 Helper.java
12/07/99  04:27p           9,303 ListAccountGUI.java
12/07/99  04:27p           6,758 ListBanksGUI.java
12/07/99  04:27p           7,518 ListBrokersGUI.java
12/07/99  04:27p        18,889 ListSharesGUI.java
12/07/99  04:27p           7,920 LoginGUI.java
12/15/99  04:00p           2,835 Makefile
12/07/99  04:27p           1,901 MatchThread.java
12/07/99  04:27p           5,657 MessageDialog.java
12/07/99  04:27p           3,255 MessageClient.java
12/07/99  04:27p           7,618 OpenAccountGUI.java
12/07/99  04:27p           2,065 OrderDetailParam.esidl
12/07/99  04:27p           3,178 OrderDetail.java
12/07/99  04:27p           2,054 ReqAuthThread.java
12/07/99  04:27p           7,236 SelectBankGUI.java
12/07/99  04:27p           9,890 SellSharesGUI.java
12/07/99  04:27p           2,277 Semaphore.java
12/07/99  04:27p           3,005 ServiceDeployer.java
12/07/99  04:27p           2,537 ShareBrokerIntf.esidl
12/07/99  04:27p           6,943 ShareBrokerIntfFinder.java
12/07/99  04:27p           6,917 ShareBrokerImpl.java
12/07/99  04:27p           2,259 ShareBrokerEventDistributor.jav
a
12/07/99  04:27p           2,048 ShareDetailParam.esidl
```

		test
12/07/99	04:27p	3,226 ShareDetail.java
12/07/99	04:27p	6,678 StartShareBrokerService.java
12/07/99	04:27p	6,346 StartTraderService.java
12/07/99	04:27p	5,204 StatusWindow.java
12/07/99	04:27p	26,825 StockTrade.java
12/07/99	04:27p	1,946 TraderQuerySpec.esidl
12/07/99	04:27p	3,929 TraderIntfFinder.java
12/07/99	04:27p	2,080 TraderIntf.esidl
12/07/99	04:27p	3,774 TraderImpl.java
12/07/99	04:27p	5,030 TraderVocabulary.java
12/07/99	04:27p	2,730 TraderService.java
12/07/99	04:27p	1,969 Transaction.esidl
12/07/99	04:27p	2,461 UserAccount.java
	55 File(s)	321,699 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example4\CVS

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/16/99	06:23p		2,891 Entries
12/16/99	06:23p		61 Repository
12/16/99	06:23p		46 Root
12/16/99	06:23p		846 Template
	6 File(s)		3,844 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\src\example5

12/06/00	04:33p	<DIR>	.
12/06/00	04:33p	<DIR>	..
12/07/99	04:27p		3,408 AccountNumberGenerator.java
12/07/99	04:27p		5,120 AddFundsGUI.java
12/07/99	04:27p		2,629 AddFundServlet.java
12/07/99	04:27p		1,932 Address.esidl
12/07/99	04:27p		3,248 AddSharesServlet.java
12/07/99	04:27p		7,505 AddSharesGUI.java
12/07/99	04:27p		11,907 AuthorizationWindow.java
12/07/99	04:27p		3,870 AuthoriseOrderServlet.java
12/07/99	04:27p		2,783 BankServiceIntf.esidl
12/07/99	04:27p		9,119 BankServiceIntfFinder.java
12/07/99	04:27p		17,380 BankServiceImpl.java
12/07/99	04:27p		4,436 BankService.java
12/07/99	04:27p		4,907 BankVocabulary.java
12/07/99	04:27p		33,346 BrokerLogic.java
12/07/99	04:27p		3,346 BuySharesServlet.java
12/07/99	04:27p		9,070 BuySharesGUI.java
12/07/99	04:27p		2,443 CancelOrderServlet.java


```

12/07/99 04:27p      test
12/07/99 04:27p      2,045 ShareDetailParam.esidl
12/07/99 04:27p      3,565 ShareDetail.java
12/07/99 04:27p      6,741 StartShareBrokerService.java
12/07/99 04:27p      5,365 StartTraderServiceServlet.java
12/07/99 04:27p      3,254 StartTraderService.java
12/07/99 04:27p      4,708 StatusWindow.java
12/07/99 04:27p      29,153 StockTrade.java
12/07/99 04:27p      1,945 TraderQuerySpec.esidl
12/07/99 04:27p      5,059 TraderVocabulary.java
12/07/99 04:27p      2,078 TraderIntf.esidl
12/07/99 04:27p      2,729 TraderService.java
12/07/99 04:27p      3,944 TraderIntfFinder.java
12/07/99 04:27p      3,614 TraderImpl.java
12/07/99 04:27p      1,986 Transaction.java
12/07/99 04:27p      2,545 UserAccount.java
      81 File(s)      403,670 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\sharebroker\s
rc\example5\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:23p      4,413 Entries
12/16/99 06:23p      61 Repository
12/16/99 06:23p      46 Root
12/16/99 06:23p      846 Template
      6 File(s)      5,366 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:27p      1,208 Carsales.ini
12/06/00 04:33p      <DIR>      config
12/06/00 04:33p      <DIR>      CVS
12/06/00 04:33p      <DIR>      html
12/07/99 04:27p      229 Makefile
12/07/99 04:27p      5,938 README
12/07/99 04:27p      126 runcore.bat
12/07/99 04:27p      472 runservers.bat
12/07/99 04:27p      5,923 sample.zone.properties
12/07/99 04:27p      10,177 sample.jserv.properties
12/06/00 04:33p      <DIR>      servlets
12/07/99 04:27p      2,892 setenv
12/07/99 04:27p      2,152 setenv.bat
12/07/99 04:27p      6 Slides.ppt
12/06/00 04:33p      <DIR>      src
      17 File(s)      29,123 bytes

```

test

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\config

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/07/99  04:27p                117 Buyer1.pr
12/07/99  04:27p                97 Buyer2.pr
12/07/99  04:27p                73 CarAd.pr
12/07/99  04:27p               105 CarBroker2.pr
12/07/99  04:27p               132 CarBroker1.pr
12/07/99  04:27p                22 co.MYCO1
12/07/99  04:27p                22 co.MYCO
12/06/00  04:33p      <DIR>      CVS
12/07/99  04:27p      1,129 input.ptml
12/07/99  04:27p        129 Payment.pr
12/07/99  04:27p       737 response.ptml
12/07/99  04:27p         99 Seller1.pr
12/07/99  04:27p         97 Seller2.pr
12/07/99  04:27p       141 SendMailServer.pr
                16 File(s)        2,900 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\config\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:23p        621 Entries
12/16/99  06:23p         55 Repository
12/16/99  06:23p         46 Root
12/16/99  06:23p        846 Template
                6 File(s)        1,568 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\CVS

```
12/06/00  04:33p      <DIR>      .
12/06/00  04:33p      <DIR>      ..
12/16/99  06:23p        495 Entries
12/16/99  06:23p         61 Entries.Log
12/16/99  06:23p         48 Repository
12/16/99  06:23p         46 Root
12/16/99  06:23p        846 Template
                7 File(s)        1,496 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\html


```

12/06/00 04:33p      <DIR>      test
12/06/00 04:33p      <DIR>      .
12/07/99 04:27p      5,634 AdvertiseToSell.html
12/07/99 04:27p      1,828 BuyerFirstScreen.html
12/07/99 04:27p      2,456 BuyerLogin2.html
12/07/99 04:27p      2,456 BuyerLogin1.html
12/07/99 04:27p      1,757 buyerops.html
12/07/99 04:27p      2,660 BuyerRegister2.html
12/07/99 04:27p      2,660 BuyerRegister1.html
12/07/99 04:27p      1,878 CarbrokerFirstScreen2.html
12/07/99 04:27p      1,876 CarbrokerFirstScreen1.html
12/07/99 04:27p      3,469 choice2.html
12/07/99 04:27p      3,469 choice1.html
12/06/00 04:33p      <DIR>      CVS
12/06/00 04:33p      <DIR>      images
12/07/99 04:27p      14,010 newcarsales.html
12/07/99 04:27p      1,824 SellerDetails.html
12/07/99 04:27p      1,888 SearchForSellers.html
12/07/99 04:27p      2,456 SellerLogin1.html
12/07/99 04:27p      1,829 SellerFirstScreen.html
12/07/99 04:27p      2,661 SellerRegister2.html
12/07/99 04:27p      2,661 SellerRegister1.html
12/07/99 04:27p      1,661 sellerops.html
12/07/99 04:27p      2,458 SellerLogin2.html
12/07/99 04:27p      1,343 thanks.html
12/07/99 04:27p      20,513 webdemo.html
26 File(s)          83,447 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\html\CVS

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:23p      1,205 Entries
12/16/99 06:23p      16 Entries.Log
12/16/99 06:23p      53 Repository
12/16/99 06:23p      46 Root
12/16/99 06:23p      846 Template
7 File(s)          2,166 bytes

```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\html\images

```

12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:27p      13,761 buy.jpg
12/07/99 04:27p      11,004 car3_anm.gif
12/07/99 04:27p      19,273 carbroker.jpg

```

```

12/06/00 04:33p      <DIR>      test
12/07/99 04:27p      CVS
12/07/99 04:27p      52  espeak.JPG
12/07/99 04:27p      2,578 happy.gif
12/07/99 04:27p      82,264 interact.jpg
12/07/99 04:27p      971  left-hand.gif
12/07/99 04:27p      971  right-hand.gif
12/07/99 04:27p      98,743 s1.jpg
12/07/99 04:27p      160,012 s10.jpg
12/07/99 04:27p      161,302 s11.jpg
12/07/99 04:27p      110,035 s2.jpg
12/07/99 04:27p      123,161 s3.jpg
12/07/99 04:27p      128,175 s4.jpg
12/07/99 04:27p      131,157 s5.jpg
12/07/99 04:27p      141,087 s6.jpg
12/07/99 04:27p      157,932 s7.jpg
12/07/99 04:27p      161,582 s8.jpg
12/07/99 04:27p      159,451 s9.jpg
12/07/99 04:27p      7,986 seller.jpg
12/07/99 04:27p      398  visal_60x38_a.gif
24 File(s)          1,671,895 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\html\images\CVS
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/16/99 06:23p      965  Entries
12/16/99 06:23p      60  Repository
12/16/99 06:23p      46  Root
12/16/99 06:23p      846  Template
6 File(s)          1,917 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\servlets
12/06/00 04:33p      <DIR>      .
12/06/00 04:33p      <DIR>      ..
12/07/99 04:27p      880  compile.bat
12/06/00 04:33p      <DIR>      CVS
12/15/99 04:00p      663  Makefile
12/07/99 04:27p      4,158 Register.java
12/07/99 04:27p      38,178 RunCarSellerClient.java
12/07/99 04:27p      46,235 RunCarBuyerClient.java
12/07/99 04:27p      13,415 RunCarBrokerService.java
9 File(s)          103,529 bytes

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\servlets\CVS

```

test

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/16/99 06:23p 326 Entries
12/16/99 06:23p 57 Repository
12/16/99 06:23p 46 Root
12/16/99 06:23p 846 Template
        6 File(s) 1,275 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\s
rc

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
12/07/99 04:27p 11,086 BrokerManager.java
12/07/99 04:27p 2,088 BuyerInfo.esidl
12/07/99 04:27p 19,355 BuyerManager.java
12/07/99 04:27p 2,416 CarAdIntf.esidl
12/07/99 04:27p 13,041 CarAdImpl.java
12/07/99 04:27p 2,902 CarBrokerIntf.esidl
12/07/99 04:27p 15,108 CarBrokerImpl.java
12/07/99 04:27p 2,416 CarDetails.esidl
12/07/99 04:27p 1,556 compile.bat
12/07/99 04:27p 2,929 Constants.java
12/06/00 04:33p <DIR> CVS
12/07/99 04:27p 2,137 DealDetails.esidl
12/15/99 04:00p 2,078 Makefile
12/07/99 04:27p 2,094 PaymentIntf.esidl
12/07/99 04:27p 6,501 PaymentServer.java
12/07/99 04:27p 3,322 PaymentImpl.java
12/07/99 04:27p 2,071 PTML.esidl
12/07/99 04:27p 2,152 SellerIntf.esidl
12/07/99 04:27p 21,491 SellerManager.java
12/07/99 04:27p 2,190 SellerInfo.esidl
12/07/99 04:27p 8,164 SellerImpl.java
12/07/99 04:27p 2,259 SendMailIntf.esidl
12/07/99 04:27p 4,043 SendMailServer.java
12/07/99 04:27p 3,714 SendMailImpl.java
12/07/99 04:27p 6,565 ServiceStopper.java
12/07/99 04:27p 2,367 Stoppable.java
12/07/99 04:27p 3,580 Util.java
        29 File(s) 147,625 bytes
```

Directory of E:\e-speak-src_991217\platform\ES\tutorial\usedcarsale\s
rc\CVS

```
12/06/00 04:33p <DIR> .
12/06/00 04:33p <DIR> ..
```

12/16/99 06:23p
12/16/99 06:23p
12/16/99 06:23p
12/16/99 06:23p
6 File(s)

test
1,369 Entries
52 Repository
46 Root
846 Template
2,313 bytes

Total Files Listed:
4189 File(s)

18,116,673 bytes
0 bytes free

008027 120800